

目 录

第一章 绪论

| | |
|------------------------------|----|
| 1.1 引言 | 1 |
| 1.2 生物进化 | 2 |
| 1.3 遗传算法 | 3 |
| 1.3.1 基础用语 | 3 |
| 1.3.2 标准遗传算法 | 5 |
| 1.4 遗传算法的特点 | 10 |
| 1.4.1 遗传算法和其它传统搜索方法的对比 | 10 |
| 1.4.2 遗传算法和若干搜索方法的亲近关系 | 13 |
| 1.4.3 遗传算法和自律分布系统的亲近关系 | 14 |
| 1.5 遗传算法的研究历史和现状 | 14 |
| 1.5.1 遗传算法的研究概况 | 14 |
| 1.5.2 遗传算法研究的新焦点 | 21 |
| 1.6 遗传算法今后研究的主要课题 | 23 |
| 参考文献 | 26 |

第二章 遗传算法的基本原理和方法

| | |
|----------------------------------|----|
| 2.1 模式定理(schemate theorem) | 28 |
| 2.1.1 模式 | 29 |
| 2.1.2 模式定理 | 30 |
| 2.2 积木块假设 | 36 |
| 2.3 骗问题 | 41 |
| 2.4 隐并行性 | 48 |
| 2.5 性能评估 | 49 |

| | | |
|--------|--------------------------|----|
| 2.6 | 编码 | 51 |
| 2.6.1 | 编码问题 | 51 |
| 2.6.2 | 编码(译码)评估规范和编码原理 | 52 |
| 2.6.3 | 编码技术 | 55 |
| 2.7 | 群体设定 | 67 |
| 2.7.1 | 初始群体设定 | 67 |
| 2.7.2 | 群体多样性 | 68 |
| 2.8 | 适应度函数 | 69 |
| 2.8.1 | 目标函数映射成适应度函数 | 70 |
| 2.8.2 | 适应度函数定标(scaling) | 71 |
| 2.8.3 | 适应度函数的设计对遗传算法的影响 | 74 |
| 2.9 | 遗传操作 | 75 |
| 2.9.1 | 选择算子 | 76 |
| 2.9.2 | 交叉算子(crossover operator) | 81 |
| 2.9.3 | 变异算子(mutation operator) | 85 |
| 2.10 | 收敛性 | 88 |
| 2.10.1 | 未成熟收敛 | 88 |
| 2.10.2 | 有限马尔柯夫链 | 90 |
| 2.10.3 | 标准遗传算法的收敛性 | 92 |
| | 参考文献 | 98 |

第三章 遗传算法与组合优化

| | | |
|-------|-------------------------------|-----|
| 3.1 | 基于遗传算法的组合优化方法 | 101 |
| 3.1.1 | 遗传算法的关键参数确定 | 102 |
| 3.1.2 | 几种流行的选择机制 | 103 |
| 3.1.3 | 适应度函数的定标 | 106 |
| 3.1.4 | 二倍体(diploidy)与显性(dominance)技术 | 107 |
| 3.1.5 | 物种形成(speciation)与小生境(niche)技术 | 111 |
| 3.2 | 函数优化(function optimization) | 117 |

| | | |
|-------|------------------------------|-----|
| 3.2.1 | 问题描述 | 117 |
| 3.2.2 | 编码与适应度函数 | 121 |
| 3.2.3 | 基本遗传算法(SGA)的搜索性能 | 122 |
| 3.2.4 | 基本遗传算法的若干变体形式的搜索性能 | 127 |
| 3.3 | 背包问题(knapsack problem) | 130 |
| 3.3.1 | 问题描述 | 130 |
| 3.3.2 | 遗传编码 | 131 |
| 3.3.3 | 适应度函数 | 131 |
| 3.3.4 | 基于基本遗传算法求解背包问题 | 132 |
| 3.3.5 | 基本遗传算法的搜索能力 | 135 |
| 3.3.6 | 基于“与/或”交叉方法求解背包问题 | 136 |
| 3.4 | 货郎担问题 | 137 |
| 3.4.1 | 编码与适应度函数 | 137 |
| 3.4.2 | 交叉策略 | 138 |
| 3.4.3 | 变异技术 | 141 |
| 3.4.4 | 选择机制和群体构成 | 142 |
| 3.4.5 | 混合 GA 技术 | 143 |
| 3.4.6 | 基于遗传算法求解 TSP 的算法实现 | 143 |
| 3.5 | 混合搜索方法 | 147 |
| 3.5.1 | 概述 | 147 |
| 3.5.2 | 启发式搜索法简介 | 148 |
| 3.5.3 | 混合遗传算法(Hybrid GA) | 151 |
| 3.5.4 | 实验与讨论 | 153 |
| 3.6 | 图的划分问题 | 158 |
| 3.6.1 | 问题描述 | 158 |
| 3.6.2 | 编码与适应度函数设计 | 158 |
| 3.6.3 | 遗传操作 | 159 |
| 3.6.4 | 实验结果 | 159 |
| | 参考文献 | 161 |

第四章 遗传算法与机器学习

| | |
|-----------------------------|-----|
| 4.1 概述 | 164 |
| 4.2 分类器系统 | 166 |
| 4.2.1 规则与消息 | 168 |
| 4.2.2 桶队算法 | 171 |
| 4.2.3 遗传算法 | 174 |
| 4.3 学习系统 LS-1 | 177 |
| 4.3.1 LS-1 与 CS-1 的区别 | 177 |
| 4.3.2 LS-1 的工作原理 | 178 |
| 4.4 基于遗传算法的概念学习系统 | 182 |
| 4.4.1 搜索空间的表示 | 184 |
| 4.4.2 遗传操作 | 186 |
| 4.4.3 执行过程 | 187 |
| 4.4.4 非标准操作 | 188 |
| 4.4.5 GABIL 系统的自适应性 | 189 |
| 4.5 小结 | 191 |
| 参考文献 | 192 |

第五章 遗传算法与并行处理

| | |
|------------------------------------|-----|
| 5.1 遗传算法固有的并行性及其并行化的困难 | 195 |
| 5.1.1 源于自然的并行性 | 195 |
| 5.1.2 遗传算法理论中的并行性 | 196 |
| 5.1.3 遗传算法在并行实现上的困难 | 197 |
| 5.2 遗传算法的并行化途径 | 198 |
| 5.2.1 主从式(master-slave)并行化方法 | 198 |
| 5.2.2 粗粒度模型 | 200 |
| 5.2.3 细粒度模型 | 201 |
| 5.3 粗粒度的孤岛模型 | 202 |

| | | |
|-------|-------------------------|-----|
| 5.3.1 | 粗粒度模型的生物学依据 | 203 |
| 5.3.2 | 粗粒度模型的研究现状 | 204 |
| 5.3.3 | 孤岛模型在 MIMD 机器上的实现 | 205 |
| 5.3.4 | 扩展的分布式遗传算法 | 211 |
| 5.4 | 细粒度的邻域模型 | 214 |
| 5.4.1 | 细粒度模型的理论基础 | 215 |
| 5.4.2 | 细粒度模型的研究现状 | 216 |
| 5.4.3 | MIMD 上的细粒度模型的实现 | 217 |
| 5.4.4 | SIMD 上的细粒度模型的实现 | 220 |
| 5.5 | 粗粒度模型与细粒度模型的性能比较 | 222 |
| 5.6 | 实现并行遗传算法的一个例子 | 224 |
| 5.6.1 | 迁入式算法 | 224 |
| 5.6.2 | 迁出式算法 | 226 |
| 5.6.3 | 扩散式算法 | 227 |
| 5.7 | LCS 的并行实现 | 229 |
| 5.7.1 | 执行系统 | 230 |
| 5.7.2 | 信用系统中的分配策略 | 232 |
| 5.7.3 | 遗传算法在 LCS 中的应用 | 232 |
| 5.7.4 | LCS 的一个 MIMD 实现 | 233 |
| 5.7.5 | LCS 在 CM 机器上的实现 | 235 |
| | 参考文献 | 237 |

第六章 神经网络、模糊集理论和进化算法

| | | |
|-------|-------------------|-----|
| 6.1 | 遗传算法与神经网络 | 241 |
| 6.1.1 | 神经网络的发展 | 241 |
| 6.1.2 | 神经网络连接权的进化 | 245 |
| 6.1.3 | 神经网络结构的进化 | 250 |
| 6.1.4 | 神经网络学习规则的进化 | 257 |
| 6.2 | 遗传算法与模糊集理论 | 258 |

| | | |
|-------|------------------------|-----|
| 6.2.1 | 基于遗传算法的模糊推理规则的优化 | 259 |
| 6.2.2 | 遗传算法在模糊模式识别中的应用 | 262 |
| 6.3 | 进化算法 | 271 |
| 6.3.1 | 引言 | 271 |
| 6.3.2 | 进化算法的总框架 | 274 |
| 6.3.3 | 遗传算法 | 275 |
| 6.3.4 | 进化规划 | 277 |
| 6.3.5 | 进化策略 | 279 |
| 6.3.6 | 交叉和变异的关系 | 282 |
| 6.3.7 | 小结 | 284 |
| | 参考文献 | 285 |

第七章 遗传算法与人工生命

| | | |
|-------|---------------------|-----|
| 7.1 | 人工生命的研究内容和方法 | 287 |
| 7.1.1 | 人工生命及其特征 | 287 |
| 7.1.2 | 人工生命研究的内容与方法 | 289 |
| 7.2 | 遗传算法与人工生命进化模型 | 292 |
| 7.3 | L系统与形态生成模型 | 295 |
| 7.3.1 | L系统与植物形态 | 295 |
| 7.3.2 | 植物的形态生成模型 | 296 |
| 7.3.3 | 讨论 | 303 |
| 7.4 | 博弈型人工生态系统 | 307 |
| 7.4.1 | 博弈与策略 | 308 |
| 7.4.2 | 博弈型生态系统 | 311 |
| 7.4.3 | 生态动力学与自组织化 | 314 |
| 7.5 | 人工生命与遗传信息处理 | 318 |
| 7.5.1 | 人类信息世界 | 319 |
| 7.5.2 | 监视遗传 | 321 |
| 7.5.3 | 遗传信息处理模型 | 322 |

| | |
|-------------------------------|-----|
| 7.5.4 基于遗传信息处理模型的人工生命合成 | 325 |
| 7.5.5 人工生命与人工智能 | 327 |
| 参考文献 | 332 |

第八章 遗传算法应用实例

| | |
|--------------------------------|-----|
| 8.1 遗传算法在图像恢复中的应用 | 334 |
| 8.1.1 引言 | 334 |
| 8.1.2 图像退化模型 | 335 |
| 8.1.3 遗传算法用于图像恢复 | 335 |
| 8.1.4 遗传算法与贝叶斯方法相结合的图像恢复 | 340 |
| 8.2 遗传算法在图像识别中的应用 | 345 |
| 8.2.1 引言 | 345 |
| 8.2.2 数学模型 | 347 |
| 8.2.3 目标函数形成 | 348 |
| 8.2.4 随机全局优化方法 | 351 |
| 8.2.5 实验结果 | 352 |
| 8.3 遗传算法在控制中的应用 | 354 |
| 8.3.1 操作序列的最优化 | 355 |
| 8.3.2 倒立摆控制 | 356 |
| 8.4 调度问题 | 358 |
| 8.4.1 车间作业调度问题 | 359 |
| 8.4.2 两种解法 | 359 |
| 8.4.3 实验 | 366 |
| 8.5 硬件进化 | 368 |
| 8.5.1 硬件进化的特点 | 369 |
| 8.5.2 硬件进化的学习方法 | 371 |
| 8.5.3 实例 | 375 |
| 参考文献 | 383 |
| 附录 A SGA 程序 | 385 |

| | | |
|------|--------------|-----|
| 附录 B | TSP 程序 | 397 |
| 附录 C | CLS 程序..... | 416 |

第一章 绪 论

遗传算法是一类借鉴生物界自然选择和自然遗传机制的随机化搜索算法,由美国 J. Holland 教授提出,其主要特点是群体搜索策略和群体中个体之间的信息交换,搜索不依赖于梯度信息。它尤其适用于处理传统搜索方法难于解决的复杂和非线性问题,可广泛用于组合优化、机器学习、自适应控制、规划设计和人工生命等领域,是 21 世纪有关智能计算中的关键技术之一。本章先从生物进化讲起,接着示例介绍简单遗传算法的具体设计方法和步骤,然后归纳出遗传算法的一般特点,最后简要介绍遗传算法的研究历史和现状以及今后将研究的主要有关课题。

1.1 引 言

生命科学科学与工程科学的相互交叉、相互渗透和相互促进是近代科学技术发展的一个显著特点,而遗传算法的蓬勃发展正体现了科学发展的这一特征和趋势。

遗传算法(Genetic Algorithm——GA),是模拟达尔文的遗传选择和自然淘汰的生物进化过程的计算模型,它是由美国 Michigan 大学的 J. Holland 教授于 1975 年首先提出的^[1]。J. Holland 教授和他的研究小组围绕遗传算法进行研究的宗旨有两个,一是抽取和解释自然系统的自适应过程,二是设计具有自然系统机理的人工系统。毫无疑问,Holland 教授的研究无论对自然系统还是对人工系统都是十分有意义的。

众所周知,在人工智能领域中,有不少问题需要在复杂而庞大的搜索空间中寻找最优解或准最优解。像货郎担问题和规划问题等

组合优化问题就是典型的例子。在求解此类问题时,若不能利用问题的固有知识来缩小搜索空间则会产生搜索的组合爆炸。因此,研究能在搜索过程中自动获取和积累有关搜索空间的知识,并自适应地控制搜索过程,从而得到最优解或准最优解的通用搜索算法一直是令人瞩目的课题。遗传算法就是这种特别有效的算法。它的主要特点是简单、通用,鲁棒性强,适用于并行分布处理,应用范围广。尽管遗传算法本身在理论和应用方法上仍有许多待进一步研究的问题,但它在组合优化问题求解、自适应控制、规划设计、机器学习和人工生命等领域的应用中已展现了其特色和魅力。

1.2 生物进化

生命自从在地球上诞生以来,就开始了漫长的生物进化历程,低级、简单的生物类型逐渐发展为高级、复杂的生物类型。这一过程已经由古生物学、胚胎学和比较解剖学等方面的研究工作所证实。生物进化的原因自古至今有着各种不同的解释,其中被人们广泛接受的是达尔文的自然选择学说。

自然选择学说认为,生物要生存下去,就必须进行生存斗争。生存斗争包括种内斗争、种间斗争以及生物跟无机环境之间的斗争三个方面。在生存斗争中,具有有利变异(mutation)的个体容易存活下来,并且有更多的机会将有利变异传给后代;具有不利变异的个体就容易被淘汰,产生后代的机会也少得多。因此,凡是在生存斗争中获胜的个体都是对环境适应性比较强的。达尔文把这种在生存斗争中适者生存,不适者淘汰的过程叫做自然选择。达尔文的自然选择学说表明,遗传和变异是决定生物进化的内在因素。遗传是指父代与子代之间,在性状上存在的相似现象。变异是指父代与子代之间,以及子代的个体之间,在性状上或多或少地存在的差异现象。在生物体内,遗传和变异的关系十分密切。一个生物体的遗传性状往往会发生变异,而变异的性状有的可以遗传。遗传能使生物性状不

断地传送给后代,因此保持了物种的特性,变异能够使生物性状发生改变,从而适应新的环境而不断地向前发展。

生物的各项生命活动都有它的物质基础,生物的遗传与变异也是这样。根据现代细胞学和遗传学的研究得知,遗传物质的主要载体是染色体(chromosome),染色体主要是由 DNA(脱氧核糖核酸)和蛋白质组成,其中 DNA 又是最主要的遗传物质。现代分子水平的遗传学的研究又进一步证明,基因(gene)是有遗传效应的片段,它储存着遗传信息,可以准确地复制,也能够发生突变,并可通过控制蛋白质的合成而控制生物的性状。生物体自身通过对基因的复制(reproduction)和交叉(crossover,即基因分离、基因自由组合和基因连锁互换)的操作使其性状的遗传得到选择和控制。同时,通过基因重组、基因变异和染色体在结构和数目上的变异产生丰富多采的变异现象。需要指出的是,根据达尔文进化论,多种多样的生物之所以能够适应环境而得以生存进化,是和上述的遗传和变异生命现象分不开的。生物的遗传特性,使生物界的物种能够保持相对的稳定;生物的变异特性,使生物个体产生新的性状,以至于形成了新的物种,推动了生物的进化和发展。

1.3 遗传算法

遗传算法是模拟前述生物进化过程的计算模型。遗传算法作为一种新的全局优化搜索算法,以其简单通用、鲁棒性强、适于并行处理以及应用范围广等显著特点,奠定了它作为 21 世纪关键智能计算之一的地位。

1.3.1 基础用语

由于遗传算法是自然遗传学和计算机科学相互结合渗透而成的新的计算方法,所以遗传算法中经常使用有关自然进化中的一些基础用语。了解这些用语对于学习和应用遗传算法是十分必要的。

如前所述,生物的遗传物质的主要载体是染色体,DNA 是其中最主要的遗传物质,而基因又是控制生物性状的遗传物质的功能单位和结构单位。复数个基因组成染色体,染色体中基因的位置称作基因座(locus),而基因所取的值又叫做等位基因(alleles)。基因和基因座决定了染色体的特征,也就决定了生物个体的性状。此外,染色体有两种相应的表示模式,即基因型(genotype)和表现型(phenotype)。所谓表现型是指生物个体所表现出来的性状,而基因型指与表现型密切相关的基因组成。同一种基因型的生物个体在不同的环境条件下可以有不同的表现型,因此表现型是基因型与环境条件相互作用的结果。在遗传算法中,染色体对应的是数据或数组,在标准的遗传算法中,这通常是由一维的串结构数据来表现的。串上各个位置对应上述的基因座,而各位置上所取的值对应上述的等位基因。遗传算法处理的是染色体,或者叫基因型个体(individuals)。一定数量的个体组成了群体(population),也叫集团。群体中个体的数目称为群体的大小(population size),也叫群体规模。而各个体对环境的适应程度叫做适应度(fitness)。

此外,执行遗传算法时包含两个必需的数据转换操作,一个是表现型到基因型的转换,另一个是基因型到表现型的转换。前者是把搜索空间中的参数或解转换成遗传空间中的染色体或个体,此过程又叫做编码(coding)操作;后者是前者的一个相反操作,叫做译码(decoding)操作。关于编码和译码操作的详细介绍将在第二章中进行。

为了便于读者在以后各章节学习遗传算法时查阅方便,将自然遗传学和人工遗传算法中所使用的基础用语的对应关系列于表 1.1 中。

表 1.1 遗传学和遗传算法中基础用语对照表

| 自然遗传学 | 人工遗传算法 |
|-----------------|--------------|
| 染色体(chromosome) | 数据、数组、位串 |
| 基因(gene) | 特质、个性、探测器、位 |
| 等位基因(allele) | 特性值 |
| 基因座(locus) | 串中位置 |
| 基因型(genotype) | 结构 |
| 表现型(phenotype) | 参数集、解码结构、候选解 |
| 遗传隐匿(epistasis) | 非线性 |

1.3.2 标准遗传算法

遗传算法是具有“生成+检测”(generate-and-test)的迭代过程的搜索算法。它的基本处理流程如图 1.1 所示。

由图 1.1 可见,遗传算法是一种群体型操作,该操作以群体中的所有个体为对象。选择(selection)、交叉(crossover)和变异(mutation)是遗传算法的 3 个主要操作算子,它们构成了所谓的遗传操作(genetic operation),使遗传算法具有了其它传统方法所没有的特性。遗传算法中包含了如下 5 个基本要素:1)参数编码;2)初始群体的设定;3)适应度函数的设计;4)遗传操作

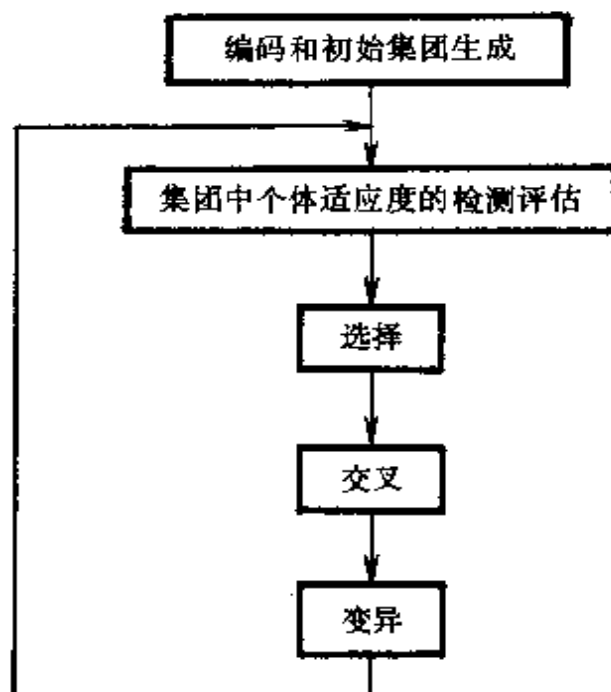


图 1.1 遗传算法的基本流程

设计；5)控制参数设定(主要是指群体大小和使用遗传操作的概率等)。这5个要素构成了遗传算法的核心内容。我们将在第二章中对它们作详细深入的论述。这里，我们仅以一个简单的函数极值求解为例，来概述遗传算法的基本概念及处理过程。

假定用遗传算法求 $f(x)=x^2$ 函数的最大值， $x \in [0,31]$ 。表 1.2 给出了用遗传算法求解此问题的计算过程和结果。

下面我们对表 1.2 中的各计算量作详细介绍。

1. 编码

由于遗传算法不能直接处理解空间的解数据，因此我们必须通过编码将它们表示成遗传空间的基因型串结构数据。在表 1.2 中，我们把变量 X 编码为 5 位长的二进制无符号整数表示形式。比如 $X=13$ 可表示为 01101 的形式。

2. 初始群体的生成

由于遗传算法的群体型操作需要，所以我们必须为遗传操作准备一个由若干初始解组成的初始群体。为了简化说明，在表 1.2 中我们取群体大小为 4，即群体由 4 个个体组成。在表 1.2 左端给出了这一初始群体的成员。需要说明的是，初始群体的每个个体都是通过随机方法产生的。初始群体也称作为进化的初始代，即第一代(first generation)。

3. 适应度评估检测

遗传算法在搜索进化过程中一般不需要其它外部信息，仅用评估函数值来评估个体或解的优劣，并作为以后遗传操作的依据。评估函数值又称作适应度(fitness)。这里，我们根据 $f(x)=x^2$ 来评估群体中各个体。显然，为了利用 $f(x)=x^2$ 这一评估函数，即适应度函数，要把基因型个体译码成表现型个体，即搜索空间中的解，比如 11000 要译码为 24。

4. 选择(selection)

选择或复制操作的目的是为了从当前群体中选出优良的个体，使它们有机会作为父代为下一代繁殖子孙。判断个体优良与否的准

表 1.2 遗传算法求 $f(x)=x^2$ 极值的计算流程

| 串编号 | 初始群体 (随机产生) ($n=4$) | X 值 (无符号 整数) | 适应度 $f(x)=x^2$ | 选择概率 $P_i = f_i / \sum f$ | 适应度 期望值 f_i / \bar{f} | 实际计 数(来 自赌轮) | 复制后 交配率 (竖线表 示交叉处) | 配对 随机 选择) | 交叉位置 (随机选择) | 新一代 群体 | X 值 | $f(x)=x^2$ |
|-----------------------------------|-----------------------------|--------------------|-------------------|------------------------------|-------------------------------|--------------------|-----------------------------|-----------------|----------------|-----------|-----|------------|
| 1 | 01101 | 13 | 169 | 0.14 | 0.58 | 1 | 0110:1 | 2 | 4 | 01100 | 12 | 144 |
| 2 | 11000 | 24 | 576 | 0.49 | 1.97 | 2 | 1100 0 | 1 | 4 | 11001 | 25 | 625 |
| 3 | 01000 | 8 | 64 | 0.06 | 0.22 | 0 | 11 000 | 4 | 2 | 11011 | 27 | 729 |
| 4 | 10011 | 19 | 361 | 0.31 | 1.23 | 1 | 10 011 | 3 | 2 | 10000 | 16 | 256 |
| 适应度总和($\sum f$) | | | 1170 | 1.00 | 4.00 | 4.0 | | | | | | 1754 |
| 平均适应度($\bar{f} = \sum f_i / n$) | | | 293 | 0.25 | 1.00 | 1.0 | | | | | | 439 |
| 最大适应度 | | | 576 | 0.49 | 1.97 | 2.0 | | | | | | 729 |

则就是各自的适应度值。显然这一操作是借用了达尔文适者生存的进化原则,即个体适应度越高,其被选择的机会就越多。选择操作实现方式很多,这里,我们采用和适应度值成比例的概率方法来进行选择的方法。具体地说,就是首先计算群体中所有个体适应度的总和($\sum f$),再计算每个个体的适应度所占的比例($f_i/\sum f$),并以此作为相应的选择概率(P_i)。表 1.2 给出了选择 4 个

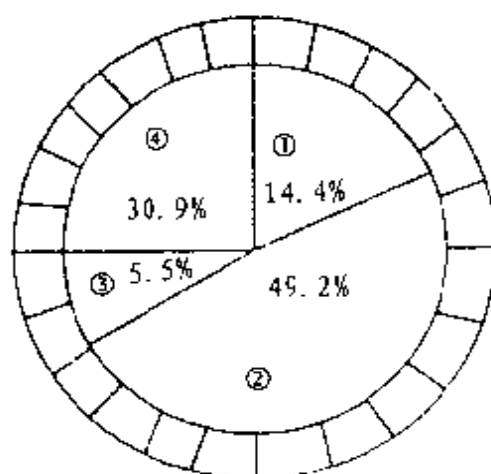


图 1.2 使用赌轮方法的选择

个体的概率,由此概率可计算出每个个体被选择的次数。我们也可以采用图 1.2 所示的赌轮(roulette wheel)方式来决定各个体的选择份数^[2]。赌轮上按前述的各个体适应按比例进行了分配。转动赌轮 4 次,我们就可以决定各自的选择份数。表 1.2 给出了相应的结果:个体 1 和个体 4 各复制 1 份,个体 2 复制 2 份,个体 3 不复制而被淘汰。这是我们所期待的结果,即最优秀的个体(个体 2,适应度为 576)获得了最多的生存繁殖机会(2 份复制),最差的个体(个体 3,适应度为 64)被淘汰。由此得到的 4 份复制送到配对库(mating pool),以备配对繁殖。

5. 交叉操作

简单的交叉(即一点交叉)可分两步进行:首先对配对库中的个体进行随机配对;其次,在配对个体中随机设定交叉处(表 1.2 中的配对库中的竖线表示交叉处),配对个体彼此交换部分信息。由表 1.2 可见,配对库中的个体 2 与个体 1 配对,交叉处为 4,通过交叉得到了两个新的个体:

$$\begin{array}{ccc}
 0110|1 & \xrightarrow{\text{交叉}} & 01100 \\
 1100|0 & & 11001
 \end{array}$$

同样,配对库中的个体 3 和个体 4 的配对交叉(交叉处为 2)得到另外两个新个体 11011 和 10000。这 4 个新个体形成了新的群体,即新一代。需要指出的是,交叉操作是遗传算法中最主要的遗传操作。由于交叉操作我们得到了新一代个体。仔细观察比较表 1.2 中新旧群体,不难发现新群体中个体适应度的平均值和最大值都有了明显提高。由于这个例子的适应度函数也就是函数 $f(x)=x^2$ 本身,所以函数值也变大了。由此可见,新群体中的个体(即解)的确是朝着我们所期望的方向进化了。当然,这个例子所用的是单变量函数,这也许不足以说明遗传算法的优点,但在本书第三章中我们将会看到,对于由几十到几百个变量组成的函数,遗传算法照样能不依靠任何搜索空间的外部知识而仅用适应度函数来指导和优化搜索的方向。

6. 变异

变异操作是按位(bit)进行的,即把某一位的内容进行变异。对于二进制编码的个体来说,若某位原为 0,则通过变异操作就变成了 1,反之亦然。变异操作同样也是随机进行的。一般而言,变异概率 P_m 都取得很小。在这个例子中,我们取 $P_m=0.001$,由于群体中共有 $20 \times 0.001=0.02$ 位可以变异,这意味群体中没有一位可变异。变异操作是十分微妙的遗传操作,它需要和交叉操作妥善配合使用,目的是挖掘群体中个体的多样性,克服有可能限于局部解的弊病。关于这一点我们将在第二章中进行充分的讨论。

在本例的模拟计算中,我们仅通过一代进化就使问题的解得到了优化,如果按图 1.1 所示的那样进行多次迭代处理,或者说群体继续不断地一代代进化下去,那么,最终一定可以得到最优解或近似最优解。

上述的遗传算法操作过程构成了标准的遗传算法,有时也叫简单遗传算法,简称 SGA(Simple GA)。SGA 的特点是:1)采用赌轮选择方法;2)随机配对;3)采用一点交叉并生成两个子个体;4)群体内

允许有相同的个体存在。

通过上述简单例子的讨论,我们不难发现,遗传算法所依赖的两个必不可少而又最重要的操作(即选择和交叉)是出乎意料的简单可行。除了随机数产生、串结构数据复制以及部分串结构数据互换外,似乎没有更多更复杂行为。读者一定会问,在这种简单而似乎有些盲目的结构数据复制和重组的操作中,遗传算法究竟干了些什么?其中是否存在某种规律或带有指导性的结论?本书的第二章将对这些问题进行深入的讨论并给予回答。

1.4 遗传算法的特点

遗传算法的特点,可以从它和传统的搜索方法的对比以及分析它和若干搜索方法与自律分布系统的亲近关系充分体现出来。

1.4.1 遗传算法和其它传统搜索方法的对比

首先让我们把它和几个主要的传统搜索方法作一简要对比,以此来看看遗传算法的鲁棒性到底强在哪里?作为一个搜索方法,它的优越性到底体现在何处?

解析方法是常用的搜索方法之一。它通常是通过求解使目标函数梯度为零的一组非线性方程来进行搜索的。一般而言,若目标函数连续可微,解的空间方程比较简单,解析法还是可以用的。但是,若方程的变量有几十或几百个时,它就无能为力了。爬山法也是常用的搜索方法,它和解析法一样都是属于寻找局部优解的方法。对于爬山法而言,只有在更好的解位于当前解附近的前提下,才能继续向优解搜索。显然这种方法对于具有单峰分布性质的解空间才能进行行之有效的搜索,并得到最优解,而对于如图 1.3 所示的多峰空间,爬山法(包括解析法)连局部最优解都很难得到。

另一种典型的搜索方法是穷举法。该方法简单易行,即在一个连续有限搜索空间或离散无限搜索空间中,计算空间中每个点的目

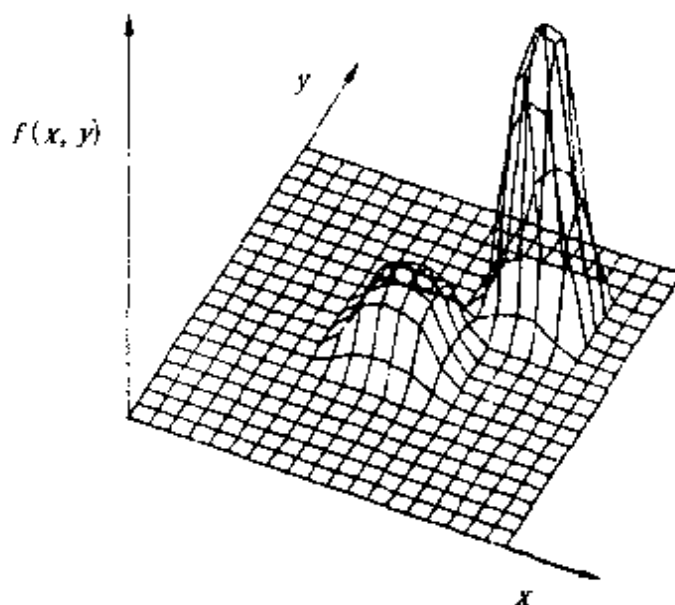


图 1.3 多峰函数

标函数值,且每次计算一个。显然,这种方法效率太低而鲁棒性不强。许多实际问题所对应的搜索空间都很大,不允许一点一点地慢慢求解。

随机搜索方法比起上述的搜索方法有所改进,是一种常用的方法,但它的搜索效率依然不高。一般而言,只有解在搜索空间中形成紧致分布时,它的搜索才有效。但这一条件在实际应用中难于满足。需要指出的是,我们必须把随机搜索(random search)方法和随机化技术(randomized technique)区分开来。遗传算法就是一个利用随机化技术来指导对一个被编码的参数空间进行高效搜索的方法。而另一个搜索方法——模拟退火(simulated annealing)方法也是利用随机化处理技术来指导对于最小能量状态的搜索。因此,随机化搜索技术并不意味着是无方向搜索,这一点是与随机搜索有所不同的。

当然,前述的几种传统的搜索方法虽然鲁棒性不强,但这些方法在一定的条件下,尤其是将它们混合使用也是有效的。不过,当面临更为复杂的问题时,必须采用像遗传算法这样更好的方法。

遗传算法具有十分顽强的鲁棒性,这是因为比起普通的优化搜

索方法，它采用了许多独特的方法和技术，归纳起来，主要有以下几个方面：

(1) 遗传算法的处理对象不是参数本身，而是对参数集进行了编码的个体。此编码操作，使得遗传算法可直接对结构对象进行操作。所谓结构对象泛指集合、序列、矩阵、树、图、链和表等各种一维或二维甚至三维结构形式的对象。这一特点，使得遗传算法具有广泛的应用领域。比如：

1) 通过对连接矩阵的操作，遗传算法可用来对神经网络或自动机的结构或参数加以优化；

2) 通过对集合的操作，遗传算法可实现对规则集合或知识库的提炼而达到高质量的机器学习目的；

3) 通过对树结构的操作，用遗传算法可得到用于分类的最佳决策树；

4) 通过对任务序列的操作，遗传算法可用于任务规划，而通过对操作序列的处理，遗传算法可自动构造顺序控制系统。

(2) 如前所述，许多传统搜索方法都是单点搜索算法，即通过一些变动规则，问题的解从搜索空间中的当前解(点)移到另一解(点)。这种点对点的搜索方法，对于多峰分布的搜索空间常常会陷于局部的某个单峰的优解。相反，遗传算法是采用同时处理群体中多个个体的方法，即同时对搜索空间中的多个解进行评估。更形象地说，遗传算法是并行地爬多个峰。这一特点使遗传算法具有较好的全局搜索性能，减少了陷于局部优解的风险。同时，这使遗传算法本身也十分易于并行化。

(3) 在标准的遗传算法中，基本上不用搜索空间的知识或其它辅助信息，而仅用适应度函数值来评估个体，并在此基础上进行遗传操作。需要着重提出的是，遗传算法的适应度函数不仅不受连续可微的约束，而且其定义域可以任意设定。对适应度函数的唯一要求是，对于输入可计算出加以比较的正的输出。遗传算法的这一特点使它的应用范围大大扩展。

(4) 遗传算法不是采用确定性规则,而是采用概率的变迁规则来指导它的搜索方向。在以后的章节中我们将会看到,遗传算法采用概率仅仅是作为一种工具来引导其搜索过程朝着搜索空间的更优化的解区域移动。因此虽然看起来它是一种盲目搜索方法,但实际上有明确的搜索方向。

上述这些具有特色的技术和方法使得遗传算法使用简单,鲁棒性强,易于并行化,从而应用范围甚广。

1.4.2 遗传算法和若干搜索方法的亲近关系

如果从更高的层次来观察遗传算法,我们不难发现它和若干搜索方法有着明显的亲近关系。分析这些关系可使我们从另一个侧面更深入地了解遗传算法的特点。

1. 遗传算法和射束搜索(beam search)方法

射束搜索方法是为了抑制搜索空间计算量的组合爆炸而提出的一种最优搜索方法。该方法预先把射束幅度定义为一个长度为 N 的开放表,在搜索的进程中仅维持 N 个最优节点,其它节点一律舍去。通过搜索,若发现有新的更好的节点,则用它把开放表中最差的节点替换掉。该搜索过程和遗传算法有一定的相似性。遗传算法中的“群体大小”相当于射束搜索中的“射束幅度”。

2. 遗传算法和单纯方法(simplex method)

单纯方法是一种直接搜索方法。它把目标函数值排序加以利用。这样,由多个端点形成的单路就可对应山的形状,然后进行爬山搜索。单纯方法的基本操作是反射(reflection)操作,且反复进行。这十分类似于遗传算法中的“交叉”操作。同时单纯方法中形成单路的端点数相当于遗传算法中的群体大小。显然,单纯方法和遗传算法在利用多点信息的全局处理上是有共同点的。

3. 遗传算法和模拟退火法

模拟退火法的最大特点是搜索中可以摆脱局部解,这是传统的爬山法所不具备的。遗传算法中的“选择”操作是以和各个体的适应

度有关的概率来进行的。因此,即使是适应度低的个体也会有被选择的机会。在这一点上它同模拟退火法十分相似。显然,通过在搜索过程中动态地控制选择概率,遗传算法可以实现模拟退火中的温度控制功能。

1.4.3 遗传算法和自律分布系统的亲近关系

所谓自律分布系统是指众多的自主分布的个体或要素,通过个体间或个体与环境间的相互作用,在群体内形成一定的秩序,并由此实现全局目标且能灵活适应环境变化。作为自律分布系统应满足如下基本条件:1)个体的自律性;2)个体间相互作用的非确定性;3)秩序的形成;4)对环境变化的适应性。在遗传算法中,个体是由具有自律性的染色体来定义特征的。遗传算法中的本质操作——交叉操作具有非确定性的相互作用。遗传算法淘汰不适应环境的个体,保留或生成能很好适应环境的个体,这种基于适应度最优的评估规范可在群体内形成秩序。同时,遗传算法通过维持群体内个体的多样性使其具有潜在的适应环境变化的能力。因此,遗传算法具备作为自律分布系统的基本条件和特征。

1.5 遗传算法的研究历史和现状

1.5.1 遗传算法的研究概况^[2]

遗传算法研究的兴起是在 80 年代末和 90 年代初期,但它的历史起源可追溯到 60 年代初期。早期的研究大多以对自然遗传系统的计算机模拟为主。早期遗传算法的研究特点是侧重于对一些复杂的操作的研究。虽然其中像自动博弈、生物系统模拟、模式识别和函数优化等给人以深刻的印象,但总的来说,这是一个无明确目标的发展时期,缺乏带有指导性的理论和计算工具的开拓。这种现象直到 70 年代中期由于 Holland 和 DeJong 的创造性研究成果的发表才得

到改观^[1,3]。当然,早期的研究成果对于遗传算法的发展仍然有一定的影响,尤其是其中一些有代表性的技术和方法已为当前的遗传算法所吸收和发展。表 1.3 列出了从 60 年代到 80 年代这一时期中遗传算法研究的发展概况。下面我们将对此作简要的说明。

在遗传算法作为搜索方法用于人工智能系统中之前,已有不少生物学家用计算机来模拟自然遗传系统。尤其是 Fraser 的模拟研究,他于 1962 年提出了和现在的遗传算法十分相似的概念和思想^[4]。但是,Fraser 和其他一些学者并未认识到自然遗传算法可以转化为人工遗传算法。而 Holland 教授及其学生不久就认识到这一转化的重要性^[1]。Holland 认为比起寻找这种或那种具体的求解问题的方法来说,开拓一种能模拟自然选择遗传机制的带有一般性的理论和方法更有意义。在这一时期,Holland 不但发现了基于适应度的人工遗传选择的基本作用,而且还对群体操作等进行了认真的研究。1965 年,他首次提出了人工遗传操作的重要性,并把这些应用于自然系统和人工系统中。

表 1.3 60 年代到 80 年代遗传算法(GA)的研究概况

| 年代 | 研究者 | 研究内容 |
|-------|---------------------|----------------------|
| 生物学 | | |
| 1967 | Rosenberg | 单细胞器官群体的进化模拟 |
| 1970 | Weinberg | 包括元 GA 的细胞群体仿真 |
| 1984 | Perry | GA 中小生境理论和种群的研究 |
| 1985 | Grosso | 带显性子群体和迁移的双倍体 GA 研究 |
| 1987 | Sannier 和 Goodman | 遗传算法调整结构以响应时空食物环境 |
| 计算机科学 | | |
| 1967 | Bagley | 采用 GA 的六子棋评估函数中的参数搜索 |
| 1979 | Raghavan 和 Birchard | 基于 GA 的聚类算法 |
| 1982 | Gerardy | 采用 GA 的概率自动机识别尝试 |

续表

| 年代 | 研究者 | 研究内容 |
|--------|-------------------------|----------------------------------|
| 1984 | Gordon | 采用 GA 的适应文本说明 |
| 1985 | Rendell | GA 搜索博弈评估函数 |
| 1987 | Raghavan 和 Agarwal | 利用 GA 的适应文本聚类 |
| 工程和运筹学 | | |
| 1981 | Goldberg | 用 SGA 的优质弹簧减振系统确认 |
| 1982 | Exter, Hicks 和 Cho | 用 SGA 的递归适应滤波器设计 |
| 1983 | Goldberg | 用 GA 对瓦斯管道的稳态和暂态优化 |
| 1985a | Davis | 用 GA 解决装箱(bin-packing)问题和图着色问题 |
| 1985b | Davis | 用 GA 解决调度问题 |
| 1985 | Davis 和 Smith | GA 用于 VLSI 布局 |
| 1985 | Fourman | 采用 GA 方法的 VLSI 布局优化 |
| 1985 | Goldberg 和 Kuo | 采用 GA 方法的瓦斯管道系统的开关状态和稳定状态的优化 |
| 1986 | Glover | 采用 GA 算法的键盘配置设计 |
| 1986 | Goldberg 和 Samtani | 采用 GA 的结构优化 |
| 1986 | Goldberg 和 Smith | SGA 求解背包问题 |
| 1986 | Minga | 用 GA 优化飞机着陆支架重量 |
| 1987 | Davis 和 Coombs | 采用 GA 和先进算子的通信网络连接规模的优化 |
| 1987 | Davis 和 Ritter | 采用模拟退火和元 GA 结合的教室调度 |
| 遗传算法 | | |
| 1962 | Holland | 使用细胞状计算机程序的适应系统 |
| 1968 | Holland | 模式理论的开发 |
| 1971 | Hollstien | 采用配对和选择规则的二维函数优化 |
| 1972 | Bosworth, Foo 和 Zeigler | 真实基因中采用复杂变异的 GA 操作 |
| 1972 | Frantz | 位置的非线性和逆转的研究 |
| 1973 | Holland | GA 和双臂盗匪(two-armed bandit)中的优化分配 |

续表

| 年代 | 研究者 | 研究内容 |
|------|----------------------------|---------------------------------|
| 1973 | Martin | 类-GA 的概率算法的理论研究 |
| 1975 | De Jong | 在 5 个函数试验中的 SGA 基本参数研究 |
| 1975 | Holland | ANAS 的出版 |
| 1977 | Mercer | 由元 GA 控制的 GA |
| 1981 | Brethke | 模式平均分析中的 Walsh 函数应用 |
| 1981 | Brindle | GA 中的选择和优势度(dominance)的研究 |
| 1983 | Pettit 和 Swigger | GA 在非稳定搜索中应用的初步探讨 |
| 1983 | Wetzel | GA 求解 TSP 问题 |
| 1984 | Mauldin | 几个对维持 SGA 中群体多样性有启发的研究 |
| 1985 | Baker | DeJong 试验中的排序选择过程的试验 |
| 1985 | Booker | 对部分匹配记录、共享和配对限制的提议 |
| 1985 | Goldberg 和 Lingle | 采用部分匹配交叉(PMX)和 O-Schema 分析的 TSP |
| 1985 | Grefenstette 和 Fitzpatrick | 用噪声函数测试 SGA |
| 1985 | Schaffer | 采用子群体的 GA 方法来实现多目标优化 |
| 1986 | Goldberg | 用边界模式内容的最大化来估计优化的群体大小 |
| 1986 | Grefenstette | 用元 GA 控制的 GA |
| 1987 | Baker | 减少选择过程中的统计误差 |
| 1987 | Bridges 和 Goldberg | 对于 L -位 GA 中的再生和交叉的扩展分析 |
| 1987 | Goldberg | 再生和交叉操作下的最小骗问题 |
| 1987 | Goldberg 和 Richardson | 采用共享函数的小生境和种群的研究 |
| 1987 | Goldberg 和 Segrest | 用有限马尔可夫链分析再生和变异 |
| 1987 | Goldberg 和 Smith | 用双倍体 GA 的非稳态函数的优化 |
| 1987 | Oliver, Smith 和 Holland | 排列重组算子的模拟和分析 |
| 1987 | Schaffer | 模式采样中选择过程效用的分析 |
| 1987 | Schaffer 和 Morishima | 串编码适应交叉试验 |
| 1987 | Whitley | GA 选择中子代测试方法的应用 |

续表

| 年代 | 研究者 | 研究内容 |
|---------------|---|--------------------------------|
| 混合技术 | | |
| 1985 | Ackley | 有 GA 性能的神经网络算法 |
| 1985 | Brady | 用遗传算子求 TSP 问题 |
| 1985 | Grefenstette et al. | 用考虑知识的遗传算子求 TSP 问题 |
| 1985 | Dolan 和 Dyer | 采用 GA 来学习神经网络拓扑 |
| 1987 | Grefenstette | 遗传搜索中采用无报酬专用信息的问题 |
| 1987 | Liepins, Hilliard, Palmer 和 Morrow | 组合问题中盲目搜索算子和贪婪搜索算子的比较 |
| 1987 | Shaefer | 全局修正适应表现技术(ARGOT) |
| 1987 | Sirag 和 Weisser | 用模拟退火控制求解 TSP 问题的遗传算子频率 |
| 1987 | Suh 和 Van Gucht | 求解 TSP 的基于知识的遗传算子 |
| 图像处理 and 模式识别 | | |
| 1970 | Cavicchio | 用于二值模式识别的检测器选择 |
| 1984 | Fitzpatrick Grefenstette 和 Van Gucht | 用 GA 方法来进行图像对准以使图像差别最小 |
| 1985 | Englander | 用于已知图像分类的检测器选择 |
| 1985 | Gillies | 用 GA 来搜索图像特征检测器 |
| 1987 | Stadnyk | 用部分匹配法的显模式类识别 |
| 并行 GA 实现 | | |
| 1976 | Bethke | 可能的并行 GA 实现的简要理论研究 |
| 1981 | Grefenstette | 若干并行 GA 实现的简单理论研究 |
| 1987 | Cohon Hegde, Martin 和 Richards | 优化线性配置的并行实现仿真 |
| 1987 | Jog 和 Van Gucht | 基于知识和并行的 GA 的组合 |
| 1987 | Pettey, Leuze 和 Grefen- stette | 采用 DeJong 测试在 Intel 硬件上实现并行 GA |
| 1987 | Suh 和 Van Gucht | 并行 GA 搜索用于 TSP 时的局部选择操作 |
| 1987 | Tanese | 在 64-NCUBE 处理器上实现并行 GA |

续表

| 年代 | 研究者 | 研究内容 |
|-------|-----------------|----------------------|
| 物理科学 | | |
| 1985 | Shaefer | 用 GA 对定势表面的非线性方程求解 |
| 社会科学 | | |
| 1985b | Reynolds | 史前猎手头目行为模型中的 GA 适配模型 |
| 1981 | Smith 和 De Jong | 用 GA 搜索调整群体迁移模型 |
| 1985a | Axelrod | 用 GA 来模拟行为规范的进化 |
| 1985b | Axelrod | 用 GA 来求解屡犯犯人处理问题 |

1967 年, Bagley 在他的论文中首次提出了遗传算法(genetic algorithm)这一术语, 并讨论了遗传算法在自动博弈中的应用[5]。他所提出的包括选择、交叉和变异的操作已与目前遗传算法中的相应操作十分接近。尤其是他对选择操作做了十分有意义的研究。他认识到, 在遗传进化过程的前期和后期, 选择概率应合适地变动。为此, 他引入了适应度定标(scaling)概念, 这是目前遗传算法中常用的技术。同时, 他也首次提出了遗传算法自我调整概念, 即把交叉和变异的概率融于染色体本身的编码中, 从而可实现算法自我调整优化。尽管 Bagley 没有对此进行计算机模拟实验, 但这些思想对于后来遗传算法的发展所起的作用是十分明显的。

在同一时期, Rosenberg 也对遗传算法进行了研究, 他的研究依然是以模拟生物进化为主, 但他在遗传操作方面提出了不少独特的设想^[6]。1970 年 Cavicchio 把遗传算法应用于模式识别中[7]。实际上他并未直接涉及到模式识别, 而仅用遗传算法设计一组用于识别的检测器。Cavicchio 对于遗传操作以及遗传算法的自我调整也做了不少有特色的研究。

Weinberg 于 1971 年发表了题为《活细胞的计算机模拟》的论文^[8]。由于他和 Rosenberg 一样注意于生物遗传的模拟, 所以他对遗

传算法的贡献有时被忽略。实际上,他提出的多层次或多级遗传算法至今仍给人以深刻的印象。

第一个把遗传算法用于函数优化的是 Hollstien^[9]。1971 年他在论文《计算机控制系统中的人工遗传自适应方法》中阐述了遗传算法用于数字反馈控制的方法。但实际上,他主要是讨论了对于二变量函数的优化问题。其中,对于优势基因控制、交叉和变异以及各种编码技术进行了深入的研究。

1975 年在遗传算法研究的历史上是十分重要的一年。这一年, Holland 出版了他的著名专著《自然系统和人工系统的适配》^[1]。该书系统地阐述了遗传算法的基本理论和方法,并提出了对遗传算法的理论研究和发展极为重要的模式理论(schemata theory)。该理论首次确认了结构重组遗传操作对于获得隐并行性的重要性。直到这时才知道遗传操作到底在干什么,为什么又干得那么出色,这对于以后陆续开发出来的遗传操作具有不可估量的指导作用。

同年,DeJong 完成了他的重要论文《遗传自适应系统的行为分析》^[3]。他在该论文中所做的研究工作可看作是遗传算法发展进程中的一个里程碑,这是因为他把 Holland 的模式理论与他的计算实验结合起来。尽管 DeJong 和 Hollstien 一样主要侧重于函数优化的应用研究,但他将选择、交叉和变异操作进一步完善和系统化,同时又提出了诸如代沟(generation gap)等新的遗传操作技术。可以认为,DeJong 的研究工作为遗传算法及其应用打下了坚实的基础,他所得出的许多结论迄今仍具有普遍的指导意义。本书第三章将对他的研究作更详细的介绍。

进入 80 年代,遗传算法迎来了兴盛发展时期,无论是理论研究还是应用研究都成了十分热门的课题。尤其是遗传算法的应用研究显得格外活跃,不但它的应用领域扩大,而且利用遗传算法进行优化和规则学习的能力也显著提高,同时产业应用方面的研究也在摸索之中。此外一些新的理论和方法在应用研究中亦得到了迅速的发展,这些无疑均给遗传算法增添了新的活力。表 1.4 给出了目前遗传算

法所涉及的主要领域。

表 1.4 遗传算法的主要应用领域

| 应用领域 | 说 明 |
|------|------------------------|
| 控制 | 瓦斯管道控制,防避导弹控制,机器人控制 |
| 规划 | 生产规划,并行机任务分配 |
| 设计 | VLSI 布局,通信网络设计,喷气发动机设计 |
| 组合优化 | TSP 问题,背包问题,图划分问题 |
| 图像处理 | 模式识别,特征抽取 |
| 信号处理 | 滤波器设计 |
| 机器人 | 路径规划 |
| 人工生命 | 生命的遗传进化 |

从表 1.4 可见,遗传算法的应用研究已从初期的组合优化求解拓展到了许多更新、更工程化的应用方面。

1.5.2 遗传算法研究的新焦点

随着应用领域的扩展,遗传算法的研究出现了几个引人注目的新动向:一是基于遗传算法的机器学习(Genetic Base Machine Learning)^[2],这一新的研究课题把遗传算法从历来离散的搜索空间的优化搜索算法扩展到具有独特的规则生成功能的崭新的机器学习算法。这一新的学习机制对于解决人工智能中知识获取和知识优化精炼的瓶颈难题带来了希望。二是遗传算法正日益和神经网络、模糊推理以及混沌理论等其它智能计算方法相互渗透和结合^[10,11],这对开拓 21 世纪中新的智能计算技术将具有重要的意义。三是并行处理的遗传算法的研究十分活跃^[12]。这一研究不仅对遗传算法本身的发展,而且对于新一代智能计算机体系结构的研究都是十分重要的。四是遗传算法和另一个称为人工生命的崭新研究领域正不断渗透。

所谓人工生命即是用计算机模拟自然界丰富多采的生命现象,其中生物的自适应、进化和免疫等现象是人工生命的重要研究对象,而遗传算法在这方面将会发挥一定的作用。五是遗传算法和进化规划(Evolution Programming, EP)以及进化策略(Evolution Strategy, ES)等进化计算理论日益结合^[13]。EP 和 ES 几乎是和遗传算法同时独立发展起来的,同遗传算法一样,它们也是模拟自然界生物进化机制的智能计算方法,既同遗传算法具有相同之处,也有各自的特点。目前,这三者之间的比较研究和彼此结合的探讨正形成热点。本书第六章 6.3 节将专门对此进行讨论。

表 1.5 ICGA91 和 PPSN92 的论文分布

ICGA91

1. 遗传算法(共 184 篇),其中:

- | | | |
|-----------------------|----------------|----------------|
| (1) 编码表示(13 篇) | (2) 遗传操作(32 篇) | (3) 选择(7 篇) |
| (4) 集群体动力学(14 篇) | (5) 环境(10 篇) | (6) 数学基础(40 篇) |
| (7) 实现技术(10 篇) | (8) 生物模型(13 篇) | (9) 应用(33 篇) |
| (10) 和 AI 相关的论文(12 篇) | | |

2. 分类系统(19 篇),其中:

- | | | |
|---------------------|---------------|---------------|
| (1) 信用分配(2 篇) | (2) 规划提取(5 篇) | (3) 并行实现(2 篇) |
| (4) 数学系统(1 篇) | (5) 建模(2 篇) | (6) 应用(5 篇) |
| (7) 和 AI 有关的论文(2 篇) | | |
-

PPSN92

1. 进化算法的理论和拓展(23 篇)
 2. 进化算法的应用(17 篇)
 3. 并行实现和生物建模(12 篇)
-

顺便提一下,随着遗传算法研究和应用的不断深入和发展,一系列以遗传算法为主题的国际会议十分活跃。从1985年开始,国际遗传算法会议,即ICGA(International Conference on Genetic Algorithm)每两年举行一次。在欧洲,从1990年开始也每隔一年举办一次类似的会议,即PPSN(Parallel Problem Solving from Nature)会议。除了遗传算法外,大部分有关ES和EP的学术论文也出现在PPSN中。另外,以遗传算法的理论基础为中心的学术会议有FOGA(Foundation of Genetic Algorithm)^[14]。它也是从1990年开始,隔年召开一次。这些国际学术会议论文集中反映了遗传算法近些年来的最新发展和动向。

表1.5给出ICGA(1991)和PPSN(1992)两次会议中的论文分布情况。

1.6 遗传算法今后研究的主要课题

综上所述,作为一种搜索算法,遗传算法的基本框架已经形成,在各种问题的求解和应用中展现了它的特点和魅力,同时也暴露出了在理论和应用技术上的许多不足和缺陷。客观地说,遗传算法尽管有各种新策略和新提案不断地被提出,但它们几乎都是针对特定问题求解而言的,对它们的评估也都是基于对比实验,其中缺乏深刻而具普遍意义的理论分析。正因为如此,遗传算法现阶段的研究重点又回到了基本理论的开拓和深化以及更通用、有效的操作技术和方法的研究上。下面概述遗传算法现阶段研究课题的几个主要方面。

1. 优化搜索方法的研究

迄今为止,优化问题的求解仍在遗传算法研究中占很大比重,诸如TSP等组合优化问题一直是遗传算法十分活跃的研究课题。据报道,遗传算法对431个城市的TSP问题的求解已取得最优解,对666个城市已可得到准优解^[15]。尽管遗传算法比其它传统搜索方法有更强的鲁棒性,但它更擅长全局搜索而局部搜索能力却不足。研

究发现,遗传算法可以用极快的速度达到最优解的 90%左右,但要达到真正的最优解则要花费很长的时间。一些对比实验还表明,如果兼顾收敛速度和解的品质两个指标,单纯的遗传算法方法未必比其它搜索方法更优越^[16]。为此,除了要进一步改进基本理论和方法外,还要采用和神经网络、模拟退火或专家系统等其它方法相结合的策略^[17]。许多研究结果表明,采用这种混合模型可有效提高遗传算法的局部搜索能力,从而进一步改善其收敛速度和解的品质。此外,探明如何能充分发挥遗传算法优越性的问题性质和类型也是十分有意义的工作。本书第三章对遗传算法与组合优化问题作了深入的讨论。

2. 学习系统的遗传算法研究

基于遗传算法的机器学习是当前遗传算法研究的一个重要侧面。其中,最引人注目的是分类系统的研究。人们期待这种学习系统可以克服传统专家系统中知识获取的瓶颈问题^[18]。但是无论是 CS-1 系统还是 LS-1 系统^[19],它们因规则匹配的需要而导致计算量和所需的存储空间都很大,而且还有句法死板以及对长规则链生成和维持能力差等缺陷。因此,基于遗传算法的学习系统现今面临的课题有:1) 适合遗传算法的高层次知识的表示;2) 更通用的基于语义的操作;3) 搜索能力的分析;4) 更有效的体现和变动环境相互作用的适应度函数的导入等。本书第四章对遗传算法与机器学习问题作了深入的讨论。

3. 生物进化与遗传算法的研究

随着分子生物学的飞跃发展,生物的超精密结构和机制正被逐渐探明,遗传算法在吸收这些知识的基础上,其本身从形式到内涵也将受到检验并有可能发生质的飞跃。实际上,从生物进化的角度看,现在的遗传算法理论框架尚处于核糖核酸(RNA)的水平,模拟从 RNA 向 DNA(脱氧核糖核酸)的进化过程和机制,从而完善和提高遗传算法的性能是值得注意的研究课题。

4. 遗传算法的并行分布处理^[12]

伴随遗传算法应用的深入发展,并行分布遗传算法及其实现的

研究也变得十分重要。遗传算法由于其群体性操作,所以本质上具有很好的并行分布处理特性。尤其是遗传算法中各个体的适应度计算(这是遗传算法中最大的计算开销)可独立进行而彼此间无需任何通信,所以并行处理效率是很高的。但是,标准的遗传算法除适应度计算外,几乎所有的遗传操作都是建立在全局统计处理的基础上,这意味着在整个进化过程中,这种全局统计处理所引起的通信开销依然是不可忽视的。因此,设计有效的并行遗传算法及相应的实现系统对于遗传算法的理论研究和应用研究都是十分重要而迫切的任务。本书第五章将对遗传算法的并行实现作深入的讨论。

5. 人工生命与遗传算法的研究

近几年来,通过计算机模拟,再现种种生命现象以达到对生命更深刻理解的人工生命的研究正在兴起。尽管目前对人工生命的含义众说不一,但它所涉及的生命现象是很清楚的,其中有生命的起源、自我增殖、自适应、遗传进化和免疫等。已有不少学者对生态系统的演变、食物链的维持以及免疫系统的进化等用遗传算法作了生动的模拟。但是实现人工生命的手段很多,遗传算法在实现人工生命中的基本地位和能力究竟如何,这是值得研究的课题。本书第七章将对遗传算法与人工生命问题作深入的讨论。

参 考 文 献

- [1] Holland J H. Adaptation in Nature and Artificial Systems. The University of Michigan Press, 1975, MIT Press, 1992
- [2] Goldberg D E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989
- [3] De Jong K A. An Analysis of the Behaviour of a Class of Genetic Adaptive Systems; [ph D Dissertation]. University of Michigan, No. 76-9381, 1975
- [4] Fraser A S. Simulation of Genetic Systems. J of Theoretical Biology, 1962, 2: 329~346
- [5] Bagley J D. The Behavior of Adaptive System which Employ Genetic and Coorelation Algorithm; [Ph D Dissertation]. University of Michigan, No. 68-7556, 1967
- [6] Rosonberg R S. Simulation of Genetic Populations with Biochemical Properties; [ph D Dissertation]. University of Michigan, No. 67-17836, 1967
- [7] Cavicchio D J. Adaptive Search Using Simulated Evolution; [ph D Dissertation]. University of Michigan, 1970
- [8] Weinberg R. Computer Simulation of a Living Cell; [ph D Dissertation]. University of Michigan, No. 71-4766, 1970
- [9] Hollstien R B. Artifical Genetic Adaptation in Computer Control Systems; [ph D Dissertation]. University of Michigan, No. 71-23773, 1971
- [10] Harp S, et al. Towards the Genetic Synthesis of Neural Networks. ICGA, 1989. 360~369
- [11] Miller G, et al. Designing Neural Networks Using Genetic Algorithm. ICGA, 1989

- [12] Stender J (Ed). Parallel Genetic Algorithms: Theory and Application. IOS Press, 1993
- [13] Hoffmeister R, Back T. Genetic Algorithms and Evolution strategies : Similarities and Differences. PPNS-1, p455
- [14] Rowllins G (Ed). Foundations of Genetic Algorithm. Morgan Kaufmann, 1991
- [15] Brawn H. On Solving Travelling Salesman Problems by Genetic Algorithms. 文献[14], 1991. 129~133
- [16] Kitano H. Empirical Studies on the Speed of Convergence of the Neural Network Training by Genetic Algorithm. Proc of AAAI-90, 1990
- [17] Powll D, Tong S, Skolnik M. EnGENEous : Domain Independent , Machine for Design Optimization. Proc of ICGA-89, 1989
- [18] Booker, Goldberg L B, Holland J H. Classifier Systems and Genetic Algorithms. Artificial Intelligence, 1989, 40: 235 ~ 282
- [19] Smith S. A Learning System Based on Genetic Adaptive Algorithms: [Ph D Dissertation]. University of Pittsburgh, 1980

第二章 遗传算法的基本原理和方法

通过前一章的介绍,我们已初步知道遗传算法是一个以适应度函数(或目标函数)为依据,通过对群体个体施加遗传操作实现群体内个体结构重组的迭代处理过程。在这一过程中,群体个体(问题的解)一代一代地得以优化并逐渐逼近最优解。遗传算法模拟的是自然界生物优胜劣汰进化过程。但是,作为一种智能搜索算法,它的本质内涵是什么?更具体地讲,遗传算法所依赖的基本遗传操作,即选择、交叉和变异算子何以能使遗传算法体现出其它算法所没有的鲁棒性、自适应性和全局优化性等特点?本章将试图回答这些问题。

如前所述,遗传算法的实现涉及 5 个主要因素:参数的编码、初始群体的设定、评估函数即适应度函数的设计、遗传操作的设计和算法控制参数的设定。本章将结合这 5 大要素详细而深入地介绍和论述这些基础理论和方法,包括模式定理、积木块假设、最小骗问题、隐并行性、性能评估、编码方法、群体设定、适应度函数、遗传操作和收敛性等课题。

2.1 模式定理(schemata theorem)^[1]

在上一章中的求 $f(x)=x^2$ 极值的实例中,我们得到了一个深刻的印象:群体中个体的适应度越高,其生存的机会就越多,而通过交叉操作,在下一代中产生了适应度更高或者说性能更好的个体。在这一过程中,虽然随机处理有助于这种优化过程,但真正的原因或机制我们尚不清楚。从第一章中的表 1.2 所描述的两代进化过程中,我们发觉,第一代的最优个体(11000)和次优个体(10011)的随机配对并在位置 2 互相交叉时,产生了新的更优的个体(11011)。若再进一

步观察,就会发现,新的个体的结构模式(11011)与其父代个体(11***和***11)的结构模式之间似乎有着一种十分有趣的联系:1)它们的模式结构有某种相似性(similarity);2)这些相似模板(similarity templates)都对应高适应值(高于群体的平均适应度)。由此我们得到这样直观感觉:遗传算法在搜索过程中一直在搜索群体中个体的某个重要的结构相似性。于是,我们得到一个新的概念——相似模板,又叫模式(schema,其复数形式为schemata)。通过它,我们将引导出遗传算法中关键的基本理论和技术。

2.1.1 模式

模式(schema)是一个描述字符串集的模板,该字符串集中的串的某些位置上存在相似性。

不失一般性,我们考虑二值字符集{0,1},由此可以产生通常的0,1字符串。现在我们增加一个符号“*”,称作“无关符”(don't care)或通配符,即“*”既可以被当作“0”,也可以被当作“1”。这样,二值字符集{0,1}就扩展为三值字符集{0,1,*},由此可以产生诸如0110、0*11**、**01*0等字符串。

定义 2.1 基于三值字符集{0,1,*}所产生的能描述具有某些结构相似性的0,1字符串集的字符串称作模式。

以长度为5的串为例,模式*0001描述了在位置2、3、4、5具有形式“0001”的所有字符串,即{00001,10001};又比如模式*1***0描述了所有在位置2为“1”及位置5为“0”的字符串,即{01000,01010,01100,01110,11000,11010,11100,11110};而模式01010描述了只有一个串的集合,即{01010}。由此可以看出,模式的概念为我们提供了一种简洁的用于描述在某些位置上具有结构相似性的0,1字符串集合的方法。这里需要强调的是,“*”只是一个描述符,而并非遗传算法中实际的运算符号,它仅仅是为了描述上的方便而引入的符号而已。

然而,模式概念的引入并不是仅仅为了描述上的方便。在引入模

式概念前,我们所看到的遗传算法是:在某一代中, N 个互不相同的串在选择、交叉、变异等遗传算子作用下产生下一代的 N 个新的互不相同的串。那么,在两代之间究竟保留了什么性质,破坏了什么性质,我们无从得知,因为我们所看到的串都是相互独立的,互不联系的。而引入模式概念后,我们看到一个串实际上隐含着多个模式(长度为 l 的串隐含着 2^l 个模式),一个模式可以隐含在多个串中,不同的串之间通过模式而相互联系。遗传算法中串的运算实质上是模式的运算。因此,通过分析模式在遗传操作下的变化,就可以了解什么性质被延续,什么性质被丢弃,从而把握遗传算法的实质,这正是模式定理所要揭示的内容。

2.1.2 模式定理

正如上一节所述,遗传算法中串的运算实质上是模式的运算。这一节我们讨论模式在选择、交叉以及变异算子作用下的变化,进而导出模式定理。

首先,我们给出模式的两个重要概念:模式阶(schema order)和定义距(defining length)。

我们知道,一个串中隐含着多个不同的模式。确切地说,长度为 l 的串,隐含着 2^l 个不同的模式,而不同的模式所匹配的串(称作模式的样本)的个数是不同的。比如模式 $011*1*$ 与模式 $0*****$ 相比,前者所匹配的串(样本)的个数比后者少,即前者的确定性高。

为了反映这种确定性的差异,我们引入模式阶概念:

定义 2.2 模式 H 中确定位置的个数称作该模式的模式阶,记作 $O(H)$ 。

比如模式 $011*1*$ 的阶数为4,而模式 $0*****$ 为1。显然,一个模式的阶数越高,其样本数就越少,因而确定性越高。

但是,模式阶并不能反映模式的所有性质。以后将看到,即使具有同阶的模式,在遗传操作下,也会有着不同的性质。为此,我们再引入定义距概念:

定义 2.3 模式 H 中第一个确定位置和最后一个确定位置之间的距离称作该模式的定义距,记作 $\delta(H)$ 。

比如模式 $011*1*$ 的定义距为 4,而模式 $0*****$ 为 0。

模式阶和定义距描述了模式的基本性质。有了这两个概念,就可以开始讨论模式在遗传操作下的变化。令 $A(t)$ 表示第 t 代中串的群体,以 $A_j(j=1,2,\dots,n)$ 表示一代中第 j 个个体串。

首先,我们探讨选择操作对模式的作用。假设在第 t 代,群体 $A(t)$ 中模式 H 所能匹配的样本数为 m ,记作 $m(H,t)$ 。在选择中,一个串是根据其适应度进行复制的。更确切地说,一个串是以概率 $P_i = f_i / \sum f_i$ 进行选择,其中 f_i 是个体 $A_i(t)$ 的适应度。假设一代中群体大小(群体中个体的总数)为 n ,且个体两两互不相同,则模式 H 在第 $t+1$ 代中的样本数 $m(H,t+1)$ 为

$$m(H,t+1) = m(H,t) \cdot n \cdot f(H) / \sum f_i \quad (2.1)$$

其中 $f(H)$ 是模式 H 所有样本的平均适应度。令群体平均适应度为 $\bar{f} = \sum f_i / n$,则有

$$m(H,t+1) = m(H,t) \cdot f(H) / \bar{f} \quad (2.2)$$

可见,模式的增长(减少),即样本数的增加(减少),依赖于模式的平均适应度与群体平均适应度之比;那些平均适应度高于群体平均适应度的模式将在下一代中得以增长;而那些平均适应度低于群体平均适应度的模式将在下一代中减少。

现在,假定模式 H 的平均适应度一直高于群体平均适应度,且设高出部分为 $c\bar{f}$, c 为常数,则有

$$m(H,t+1) = m(H,t) \cdot (\bar{f} + c\bar{f}) / \bar{f} = m(H,t) \cdot (1+c) \quad (2.3)$$

假设从 $t=0$ 开始, c 保持为常值,则有

$$m(H,t+1) = m(H,0) \cdot (1+c)^t \quad (2.4)$$

可见,在选择算子作用下,平均适应度高于群体平均适应度的模式将呈指数级增长;而平均适应度低于群体平均适应度的模式将呈指数

级减少。

以上,我们讨论了模式在选择中的变化。然而仅仅有选择操作,并不能产生新的个体,即不能对搜索空间中新的区域进行搜索,因此引入了交叉操作。下面讨论模式在交叉算子作用下所发生的变化,这里我们只考虑单点交叉的情况。

为了讨论方便,不妨考虑一个长度为 6 的串以及隐含其中的两个模式:

$$\begin{aligned} A &= 0 \ 1 \ 0 \ 1 \ 1 \ 0 \\ H_1 &= * \ 1 \ * \ * \ * \ 0 \\ H_2 &= * \ * \ * \ 1 \ 1 \ * \end{aligned}$$

假定 A 被选中进行交叉,而交叉点等概率产生,即交叉点落在 1、2、3、4、5 的概率相同。这里不妨假设交叉点为 3,即交叉发生在位置 3 和位置 4 之间,并以分隔符“|”表示交叉位置,即有

$$\begin{aligned} A &= 0 \ 1 \ 0 \ | \ 1 \ 1 \ 0 \\ H_1 &= * \ 1 \ * \ | \ * \ * \ 0 \\ H_2 &= * \ * \ * \ | \ 1 \ 1 \ * \end{aligned}$$

在这种情况下,除了与 A 进行交叉的串(称作配偶)在确定位(比如位置 2、6)相同(这种可能性我们暂且不考虑)外,模式 H_1 将遭到破坏,因为位置 2 的“1”和位置 6 的“0”在交叉产生的子代个体中将被替代为不同的值。比如 A 的配偶为 $A' = 101001$,则产生的后代为 $A_1 = 010001$, $A_2 = 101110$,都不是 H_1 的样本,即发生交叉后,模式 H_1 丢失了。而相同情况下, H_2 却依然存在,因为不论 A 的配偶为任何串, H_2 中确定位 4 的“1”和确定位 5 的“1”都将一块传入子代。可能有的读者会问,假若交叉点在位置 4, H_2 不也会遭到破坏吗? 不错!但有一点可以看出,由于交叉点等概率产生,模式 H_1 遭破坏的概率(在位置 2、3、4、5 交叉都遭破坏)大大超过模式 H_2 (只在位置 4 交叉遭破坏),即 H_2 的“生命力”要强于 H_1 。

现在定量地讨论一下。我们注意到模式 H_1 的定义距为 4,那么交叉点在 $6 - 1 = 5$ 个位置随机产生时, H_1 遭破坏的概率为 $P_1 =$

$\delta(H_1)/(l-1) = 4/5$, 换句话说, 其生存概率为 $1/5$; 而模式 H_2 的定义距为 1, 则 H_2 遭破坏的概率为 $P_d = \delta(H_2)/(l-1) = 1/5$, 即生存概率为 $4/5$ 。

更一般地讲, 模式 H 只有当交叉点落在定义距之外才能生存。在简单交叉(单点交叉)下的 H 的生存概率 $P_s = 1 - \delta(H)/(l-1)$ 。而交叉本身也是以一定的概率 P_c 发生的, 所以模式 H 的生存概率为

$$P_s = 1 - P_c \cdot \delta(H)/(l-1) \quad (2.5)$$

现在我们考虑先前暂且忽略的可能性, 即交叉发生在定义距内, 模式 H 不被破坏的可能性。在前面的例子中, 若 A 的配偶在位置 2、6 上有一位与 A 相同, 则 H_1 将被保留。考虑到这一点, 式(2.5)给出的生存概率只是一个下界, 即有

$$P_s \geq 1 - P_c \cdot \delta(H)/(l-1) \quad (2.6)$$

式(2.6)描述了模式在交叉算子作用下的生存概率。现在考虑模式 H 在选择和交叉算子的共同作用下的变化。参照式(2.2)和式(2.6), 则有

$$m(H, t+1) \geq m(H, t) \cdot (f(H)/\bar{f}) \cdot [1 - P_c \cdot \delta(H)/(l-1)] \quad (2.7)$$

由式(2.7)可以看出, 在选择和交叉算子的共同作用下, 模式的增长(减少)取决于两个因素: 1) 模式的平均适应度是否高于群体平均适应度; 2) 模式是否具有较短的定义距。显然, 那些平均适应度高于群体平均适应度、具有短定义距的模式将呈指数级增长。

最后考虑变异操作。假定串的某一位置发生改变的概率为 P_m , 则该位置不变的概率为 $1 - P_m$, 而模式 H 在变异算子作用下若要不受破坏, 则其中所有的确定位置(为“0”或“1”的位)必须保持不变。因此模式 H 保持不变的概率为 $(1 - P_m)O(H)$, 其中 $O(H)$ 为模式 H 的阶数。当 $P_m \ll 1$ 时, 模式 H 在变异算子作用下的生存概率为

$$P_s = (1 - P_m)O(H) \approx 1 - O(H) \cdot P_m \quad (2.8)$$

综上所述, 模式 H 在遗传算子选择、交叉和变异的共同作用下,

其子代的样本数为

$$m(H, t+1) \geq m(H, t) \cdot (f(H)/\bar{f}) \cdot [1 - P_c \cdot \delta(H)/(l-1) - O(H) \cdot P_m] \quad (2.9)$$

式(2.9)忽略了极小项($P_c \cdot \delta(H)/(l-1) \cdot O(H) \cdot P_m$)。通过式(2.9),我们就可以给出以下模式定理:

定理 2.1(模式定理) 在遗传算子选择、交叉和变异的作用下,具有低阶、短定义距以及平均适应度高于群体平均适应度的模式在子代中将得以指数级增长。

模式定理是遗传算法的理论基础,其意义是深远的。为了很好地理解该定理,下面我们通过一个实例来观察一下遗传算法对模式的处理情况。采用第一章中的求 $f(x)=x^2$ 极值的实例。假设 x 的取值范围为 $[0, 31]$, 则 x 可编码为 5 位二进制数。给定 4 个初始串,然后施加选择、交叉、变异操作,有关的过程由表 2.1 表示。

让我们考察 3 个模式: $H_1=1****$ 、 $H_2=*10***$ 和 $H_3=1***0$ 的运行情况。首先来看 H_1 在遗传操作下的变化。在选择阶段,各串按照其适应度在群体适应度中所占的比例进行复制(比例复制法)。观察表 2.1,可以看出,由于串 2 的适应度大,其复制概率高,经过选择被复制成 2 份,而串 3 则丢失了。注意到串 2、4 都是 H_1 的样本,这样通过选择后, H_1 的样本增至为 3。由模式定理,理论上应有 $m \cdot f(H)/\bar{f}$ 个样本,其中 $f(H)=(576+361)/2=468.5$, 而 \bar{f} 为 293, 则 H_1 在选择算子作用下应有 $2 \cdot 468.5/293=3.2 \approx 3$ 个样本,与实际相符。进一步可以看到,由于 H_1 的定义距为 0,交叉操作对 H_1 的样本没有任何影响。而对于变异操作,在变异概率为 $P_m=0.001$ 的情况下, H_1 的 3 个样本遭破坏的概率为 $3 \cdot 0.001=0.003$, 可以认为不发生变异。事实上,我们看到子代中有 3 个 H_1 的样本 11001、11011、10000, H_1 的样本数的确如模式定理所预计的呈指数级增长。

那么模式 H_2 、 H_3 呢? H_2 的初始样本数为 2,在选择算子作用下,样本数仍为 2。这与模式定理所预计的 $2 \cdot 320/293=2.18 \approx 2$ —

表 2.1 遗传算法对模式的处理

| 串处理 | | | | | | | | | | | |
|-------|----------------|-----|-------------------|----------------------|-----------------------|--------------------|------------------------|-----|-------|------------|-----------------|
| 串号 | 初始群体 (随机产生) | X 值 | $f(x)$ x^2 | $\frac{f_i}{\sum f}$ | 期望个数 f_i/\bar{f} | 实际个 数(比例 复制) | 交叉情况 (‘1’-- 交叉点) | 配偶 | 交叉点 | 新群体 X 值 | $f(x)$ x^2 |
| 1 | 01101 | 13 | 169 | 0.14 | 0.58 | 1 | 0110 1 | 2 | 4 | 01100 | 144 |
| 2 | 11000 | 24 | 576 | 0.49 | 1.97 | 2 | 1100 0 | 1 | 4 | 11001 | 625 |
| 3 | 01000 | 8 | 64 | 0.06 | 0.22 | 0 | 11,000 | 4 | 2 | 11001 | 729 |
| 4 | 10011 | 19 | 361 | 0.31 | 1.23 | 1 | 10 011 | 3 | 2 | 10000 | 256 |
| 总和 | | | 1170 | 1.00 | 4.00 | 4 | | | | | 1754 |
| 平均 | | | 293 | 0.25 | 1.00 | 1 | | | | | 439 |
| 最大 | | | 576 | 0.49 | 1.97 | 2 | | | | | 729 |
| 模式处理 | | | | | | | | | | | |
| 选择前 | | | 选择后 | | | 下一代 | | | | | |
| 串代表 | | | 模式平均适应度 $f(H)$ | | | 期望个数 | 实际个数 | 串代表 | 期望个数 | 实际个数 | 串代表 |
| H_1 | 1 * * * * | 2,4 | 469 | 3.20 | 3 | 2,3,4 | 3.20 | 3 | 2,3,4 | | |
| H_2 | * 1 0 * * | 2,3 | 320 | 2.18 | 2 | 2,3 | 1.64 | 2 | 2,3 | | |
| H_3 | 1 * * * 0 | 2 | 576 | 1.97 | 2 | 2,3 | 0.0 | 1 | 4 | | |

致。而模式 H_3 的初始样本数为 1, 预期值为 $1 \cdot 576/293 = 1.97 \approx 2$ 与实际值相符。下面来看交叉操作。尽管 H_2 和 H_3 的阶数相同, 选择后的样本数也相同, 但由于两个模式的定义距不同, 在交叉算子作用下的结果也不同。对于 H_2 , 由模式定理, 其预期值应为 $m(H_2, t+1) = 2.18 \cdot (1 - 1/4) = 2.18 \cdot 0.75 = 1.64 \approx 2$, 实际上, 最终 H_2 的两个样本都得以保留。而对于 H_3 , 其预期值为 $m(H_3, t+1) = 1.97 \cdot (1 - 4/4) = 0$, 但事实上, H_3 有一个样本得以保留(10000)。这似乎与模式定理不相符, 但应该强调的是, 模式定理给出的只是一个下界, H_3 之所以有一个样本能生存, 是因为 H_3 的样本 11000 的配偶 10011 在 H_3 的确定位(位置 1,5)并不互补。尽管如此, 我们也可以看出, 由于 H_2 具有较短的定义距, 其生命力强于 H_3 。

以上通过一个实例证实了模式定理的正确性。在遗传算子作用下, 低阶、短距、高平均适应度的模式以指数级增长。但有一个问题还有待回答: 为什么这样一种性质就有利于找到全局最优解呢?

统计确定理论中的双角子机问题表明: 要获得最优的可行解, 则必须保证较优解的样本数呈指数级增长^[3]。而模式定理保证了较优的模式(遗传算法的较优解)的样本数呈指数级增长, 从而给出了遗传算法的理论基础。Holland 指出了遗传算法是一个寻找可行解的可实现的优化过程^[1,3]。

2.2 积木块假设

由上一节的模式定理可知, 具有低阶、短定义距以及平均适应度高于群体平均适应度的模式在子代中将以指数级增长。这类模式在遗传算法中非常重要, 我们给它们一个特别的名字——积木块 (building block)。

定义 2.4 具有低阶、短定义距以及高适应度的模式称作积木块。

正如搭积木一样, 这些“好”的模式在遗传操作下相互拼搭、结

合,产生适应度更高的串,从而找到更优的可行解,这正是积木块假设所揭示的内容。

假设 2.1 [积木块假设(building block hypothesis)] 低阶、短距、高平均适应度的模式(积木块)在遗传算子作用下,相互结合,能生成高阶、长距、高平均适应度的模式,可最终生成全局最优解。

上一节的模式定理保证了较优的模式(遗传算法的较优解)的样本数呈指数级增长,从而满足了寻找最优解的必要条件,即遗传算法存在着寻找到全局最优解的可能性。而本节的积木块假设则指出,遗传算法具备寻找到全局最优解的能力,即积木块在遗传算子作用下,能生成高阶、长距、高平均适应度的模式,最终生成全局最优解。

然而,遗憾的是上述结论并没有得到证明,正因为如此才被称为假设,而非定理。目前已经有大量的实践证据支持这一假设,从 20 多年前 Bagley 和 Rosenberg 的两篇开创性的文章^[4,5]到最近大量的遗传算法应用实例都表明,积木块假设在许多领域都获得了成功,例如平滑多峰问题,带干扰多峰问题以及组合优化问题等等。尽管大量的证据并不等于理论证明,但至少可以肯定,对多数经常碰到的问题,遗传算法都是适用的。

最近, Bethke^[6]的研究似乎为这个问题的解决带来了希望。Bethke 采用 Walsh 函数和一种巧妙的模式变换,提出了一种采用 Walsh 系数计算模式平均适应度的有效分析方法,这使得在一些特定的适应度函数和编码方式下,可以判定积木块通过相互组合是否会产生最优解或接近最优解。Holland^[7]把 Bethke 的方法推广到了当群体非均匀分布时的模式平均适应度分析上。

下面我们介绍 Walsh 模式分析^[8]。

首先引入“分割”概念。一个分割是 $\{d, *\}^l$ 的一个串,分割的长度是最左非 * 位到最右非 * 位的距离,分割的阶是串中非 * 位的个数。一个阶为 n 的分割将搜索空间分成了 2^n 个区域,每个区域相应于一个模式。将分割中的 * 换成 0, d 换成 1,得到一个二进制数,作为这个分割的序号。例如, $*dd*$ 的阶是 2,将搜索空间分成 $2^2=4$

个区域,相应的模式分别为 * 00 * , * 01 * , * 10 * , * 11 * 。这个分割的序号是 0110=6。

Walsh 函数是一族完全正交的函数,它的值域只有两个元素: +1和-1。Bethke 使用离散的 Walsh 函数,为定义在 $\{0,1\}^l$ 上的实值函数构造了一个正交基。离散的 Walsh 函数把二值串映像到 $\{1,-1\}$ 。每个 Walsh 函数对应搜索空间的一个分割,对应第 j 个分割的 Walsh 函数定义如下:

$$\Psi_j(x) = \begin{cases} 1 & \text{如果 } x \wedge j \text{ 有偶数个“1”} \\ -1 & \text{其它} \end{cases}$$

这里 x 和 j 都是二进制表示,“ \wedge ”是按位与, j 称为 Walsh 函数的阶。

设 $F(x)$ 是定义在 $\{0,1\}^l$ 上的实值函数,则有 Walsh 变换:

$$\omega_j = \frac{1}{2^l} \sum_{x=0}^{2^l-1} F(X) \Psi_j(X) \quad (2.10)$$

ω_j 称为函数 $F(x)$ 的 Walsh 系数,

$$F(X) = \sum_{j=0}^{2^l-1} \omega_j \Psi_j(X) \quad (2.11)$$

称作 $F(x)$ 的 Walsh 多项式表示,它将 $F(x)$ 表示成 Walsh 系数求和序列。这个求和过程 逐步逼近 $F(x)$,每个部分和都是对 $F(x)$ 的一个估值,越长的部分和就越接近 $F(x)$ 的真实值。求和开始时, $j=0$,

$\Psi_j(x) \equiv 1, F(x) = \omega_0 = \frac{1}{2^l} \sum_{x=0}^{2^l-1} F(x)$,它是 $F(x)$ 的平均值,记为 \bar{f} ,这是对 $F(x)$ 的最粗糙的估计。随着求和的继续, ω_j 或加或减(取决于 $\Psi_j(X)$ 的正负),不断对 $F(x)$ 做出修正,而每次的修正值就是 Walsh 系数,最终求得 $F(x)$ 的真实值。

每个分割 j 有一个 Walsh 系数 ω_j 与之对应。将 ω_j 理解为一个差值,直观意义就是分割中每个模式的真实平均值与低阶 Walsh 系数所给的估计之间的偏差。对同一分割中的每个模式,这个值是相同的。

Walsh 变换与模式有着密切的联系, Walsh 模式变换系统地描述了这种联系。前面, 对于给定的 x , 使用 Walsh 系数, 渐近地估计了 $F(x)$ 。Bethke 用相同的方法计算模式的平均强度。设 H 是一个模式, 则

$$\mu(H) = \sum_{j: H \in j} \omega_j \Psi_j(H) \quad (2.12)$$

称为 Walsh 模式变换。这里, 分割 j 包孕模式 H (记作 $H \in j$), 当且仅当 j 包含模式 H' , 使得 $H' \supseteq H$ 。例如, 三位模式 $10*$ 被 4 个分割包孕: $dd*$, $*d*$, $d**$ 和 $***$, 相应的序号为 110, 010, 100 和 000。不难看出, j 包孕 H 当且仅当 j 中每一个有定义的位在 H 均有定义。

Walsh 模式变换把一个模式 H 的强度表示为一些阶数逐渐增高的 Walsh 系数的和, Walsh 系数是一个修正项, 修正包含 H 的低阶模式所给出的估计值。不同的是, Walsh 变换求和时要考虑所有 2^l 个分割的 Walsh 系数, 但 Walsh 模式变换只对那些包孕模式 H 的分割的 Walsh 系数求和。

Walsh 模式变换的求和顺序, 基本反映了遗传算法估计模式平均值的过程。我们可以把遗传算法的群体看做模式的抽样, 算法用群体中个体的强度估计模式的强度。显然, 群体中低阶模式的样本数多于高阶模式的样本数, 因此, 低阶模式比高阶模式更早获得精确估计。遗传算法对一个给定模式 H 的估计可以看做是一个渐近的细化过程: 开始时, H 的估计值是基于包含 H 的低阶模式的信息, 逐渐用包含 H 的高阶模式的信息来精确 H 的估计值。类似的, 上述和式代表了模式 H 的真实值的渐近估计。 ω_0 是群体平均 (相当于模式 $***$ 的平均值), 高阶的 Walsh 代表了高阶修正。

因此, 可以用 Walsh 系数描述遗传算法操作: 遗传算法渐近地估计 Walsh 系数, 并且引导搜索朝着 Walsh 系数较大的分割中使 Walsh 函数取正值的模式进行, 就是说, 遗传算法是 Walsh 系数求和过程中的贪心。

Bethke 指出,如果一个函数的 Walsh 系数随着 j (相应分割的序号)的阶和长度的增长而迅速降低,即重要的 Walsh 系数是与短的低阶分割相联系的,则这个函数容易用遗传算法求解,称为 GA 易(GA easy)。这时,通过估计低阶模式的平均值就可以发现全局最优点。这样的模式在群体中所占比例比高阶长模式更容易增长;“低阶”意味着它的样本较多,“短”使得它不容易被破坏。对遗传算法而言,估计它们的平均值要比估计高阶长模式容易得多。因此,Bethke 得出结论^[6],由于遗传算法难于对高阶分割中的高阶模式做出好的估计,在其它条件都相同的情况下,如果一个函数的 Walsh 分解中高阶分割对应了较重要(数值较大)的 Walsh 系数,则该函数难于用遗传算法求解,并称之为 GA-难(GA-hard)。

从模式理论的角度来看,高阶系数小,意味着低阶模式所给出的估计误差小,信息准确,低阶积木块组合即可得到高阶积木块,满足积木块假设,所以,Walsh 模式分析和模式理论在划分 GA-难、GA-易时,实际上是殊途同归的。

对于判定一个函数是否 GA-难,Bethke 的结论并不适用,因为大部分遗传算法应用中的目标函数都不容易表示成 Walsh 多项式;即使容易表示,计算 Walsh 系数需要对搜索空间的每个点计算 $F(x)$,这对大多数函数是不可能的,而且,如果计算了 $F(x)$ 的所有值,就知道了其最优解,而没必要再用遗传算法求解了,分析就失去了意义。

但是 Walsh 分析是一个强有力的理论工具。Bethke 采用此方法已经产生了一些会使简单三算子遗传算法误入歧途的测试情况,我们称这些编码函数的组合是 GA-骗的(GA-deceptive)。这些结果告诉我们,GA-骗的函数和编码有包含孤立的最佳点的倾向,即最好的点可能被最坏点所包围。但我们真正遇到的很多函数均不含此情况。实际的情况是,函数编码组合中常存在着某种规则性,它们都可由重组积木块进行开拓之。重要的是,我们要记住,简单遗传算法依赖于为寻找最佳解点的积木块的组合,如果这些积木块组合因所使用的

编码和函数本身而被弄错,那么问题欲达到最佳解可能需要很长的时间。

2.3 骗问题

我们已经知道,遗传算法适用于大多数经常碰到的问题,但也存在着一些遗传算法难以解决的问题,即最终的搜索往往偏离全局最优解,这类问题被称作 GA-难的。这一节我们探讨所谓的骗问题(deceptive problem),即构造一个问题,给定一些带欺骗性质的初始条件,“迷惑”遗传算法,使其偏离全局最优解。为此,要最大限度地违背积木块假设,即使得低阶、短距、高于平均适应度的模式生成局部最优点。由模式理论,一个问题能否用遗传算法有效地求解,取决于问题编码是否满足积木块假设,满足者用遗传算法求解效率较高,不满足者效率较低,甚至找不到满意解。然而,尽管我们试图“迷惑”遗传算法,但这类骗问题却往往不是 GA-难的,即通常不会偏离全局最优解。

以下描述最小骗问题(minimal deceptive problem)。这里所谓“最小”是指为了造成骗局所需设置的最小的问题规模(即阶数)。

假设有一个由 4 个阶数为 2、有 2 个确定位置的模式构成的集合,各模式具有如下的适应度:

| | | | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|---|---|---------|
| (1) | * | * | * | 0 | * | * | * | * | * | 0 | * | $f(00)$ |
| (2) | * | * | * | 0 | * | * | * | * | * | 1 | * | $f(01)$ |
| (3) | * | * | * | 1 | * | * | * | * | * | 0 | * | $f(10)$ |
| (4) | * | * | * | 1 | * | * | * | * | * | 1 | * | $f(11)$ |
| $\{\leftarrow \delta(H) \rightarrow\}$ | | | | | | | | | | | | |

其中 $f(00)$, $f(01)$, $f(10)$, $f(11)$ 是各模式的平均适应度,并假定为常值。我们不妨假定 $f(11)$ 是全局最优值,即有

$$f(11) > f(00), f(11) > f(01), f(11) > f(10) \quad (2.13)$$

现在,设法引入迷惑遗传算法的条件。考虑 4 个一阶模式的适应度,

即 $f(*0)$ 、 $f(*1)$ 、 $f(0*)$ 以及 $f(1*)$ 。一阶模式的适应度等于其包含的所有二阶模式的适应度的平均值, 即有

$$f(*0) = [f(00) + f(10)]/2 \quad (2.14)$$

$$f(*1) = [f(01) + f(11)]/2 \quad (2.15)$$

$$f(0*) = [f(00) + f(01)]/2 \quad (2.16)$$

$$f(1*) = [f(10) + f(11)]/2 \quad (2.17)$$

令包含全局最优解 $f(11)$ 的一阶模式的适应度小于不包含最优解的一阶模式的适应度, 从数学上讲, 就是

$$f(0*) > f(1*) \quad (2.18)$$

$$f(*0) > f(*1) \quad (2.19)$$

由式(2.14)~式(2.17)有

$$[f(00) + f(01)]/2 > [f(10) + f(11)]/2 \quad (2.20)$$

$$[f(00) + f(10)]/2 > [f(01) + f(11)]/2 \quad (2.21)$$

式(2.20), (2.21)给出了所谓的“骗”条件, 下面我们将看到骗条件会迷惑遗传算法, 使得遗传算法偏离全局最优值 $f(11)$ 。同时可以看出, 式(2.20)和式(2.21)并不能同时成立, 否则就有 $f(00) > f(11)$, 从而违背了 $f(11)$ 是全局最优值的假定。不失一般性, 不妨假定式(2.20)成立。由此, 通过一个全局条件($f(11)$ 为全局最优值)和一个“骗”条件($f(0*) > f(1*)$), 就确定了一个骗问题。

将上述各适应度值按 $f(00)$ 进行归一化如下:

$$r = f(11)/f(00), c = f(01)/f(00), c' = f(10)/f(00) \quad (2.22)$$

则全局条件(2.13)可以表示为

$$r > 1, r > c, r > c' \quad (2.23)$$

则“骗”条件(2.20)可以表示为

$$r < 1 + c - c' \quad (2.24)$$

由式(2.23)和(2.24), 可以推出:

$$c' < 1, c' < c \quad (2.25)$$

由式(2.25)可以看出, 存在着如下两类骗问题:

类型 I : $f(01) > f(00)$ ($c > 1$)

类型 II : $f(00) \geq f(01)$ ($c \leq 1$)

图 2.1 和图 2.2 给出这两类问题的图示, 其中适应度是两个自

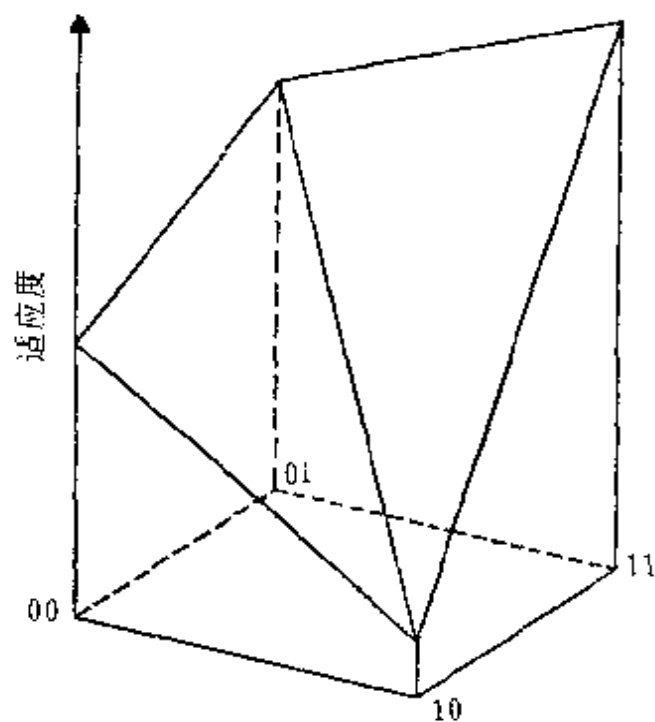


图 2.1 类型 I 最小偏差问题 ($f(01) > f(00)$)

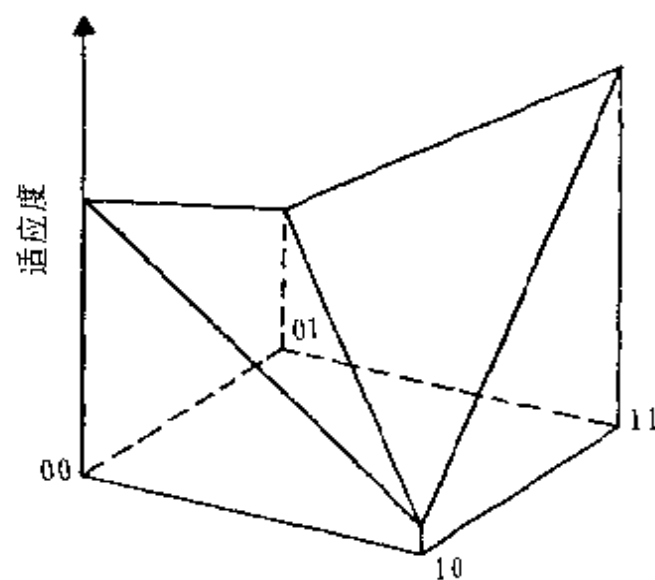


图 2.2 类型 II 最小偏差问题 ($f(00) \geq f(01)$)

变量(确定位 1、2)的函数。由此,我们定义了两类骗问题。通过观察不难发现,一阶问题(仅有一个确定位)中是不可能存在骗问题的,因为无法找到与全局条件相适应的骗条件,因此上述 2 阶骗问题是可能存在的最小骗问题。

现在开始对最小骗问题中的模式进行分析。为了达到“欺骗”目的(使遗传算法偏离全局最优解),应该使得具有全局最优值的模式在遗传算子作用下减少,则由模式定理应有:

$$\frac{f(11)}{\bar{f}} \cdot [1 - P_c \cdot \delta(11)/(l - 1)] \leq 1$$

这里假设 $P_m=0$ 。

我们已经知道,模式定理所得到的生存概率 P_c 只是一个下界。因为在交叉算子作用下,即使交叉点落在定义距内,模式也不一定会遭到破坏,这取决于配偶的情况。在上述骗问题中,00 与 01 交叉产生 01 和 00,父模式并未发生丢失。只有当一个模式确定位置与其配偶互为补数时(二进制串),才会发生丢失。比如 00 和 11 交叉产生 01 和 10;同理,01 与 10 交叉产生 11 和 00。该问题中各模式相互交叉的结果由表 2.2 表示,其中 S 表示子代与父代完全相同。

| 表 2.2 | | 2-位问题各模式相互交叉结果 | | | |
|-------|--|----------------|----|----|----|
| X | | 00 | 01 | 10 | 11 |
| 00 | | S | S | S | 01 |
| | | | | | 10 |
| 01 | | S | S | 00 | S |
| | | | | 11 | |
| 10 | | S | 00 | S | S |
| | | | 11 | | |
| 11 | | 01 | S | S | S |
| | | 10 | | | |

由表 2.2 可以看到,互补的模式(确定位互补)在交叉算子作用下遭到破坏,然而另一对互补的模式通过交叉又会重新产生遭破坏

的模式。通过表 2.2, 我们可以更准确地描述骗问题中 4 个竞争模式在子代中的期望样本数, 其中必须考虑到一对模式在交叉后发生丢失, 又会在另一对模式交叉后重新出现这个因素。假定在简单选择(按适应度比例选择)、简单交叉(单点交叉)以及等概率配对情况下, 则有

$$P_{11}^{t+1} = P_{11}^t \cdot \frac{f(11)}{\bar{f}} \cdot [1 - P_c \cdot \frac{f(00)}{\bar{f}} \cdot P_{00}^t] + P_c \cdot \frac{f(01) \cdot f(10)}{\bar{f}^2} \cdot P_{01}^t \cdot P_{10}^t \quad (2.26)$$

$$P_{10}^{t+1} = P_{10}^t \cdot \frac{f(10)}{\bar{f}} \cdot [1 - P_c \cdot \frac{f(01)}{\bar{f}} \cdot P_{01}^t] + P_c \cdot \frac{f(00) \cdot f(11)}{\bar{f}^2} \cdot P_{00}^t \cdot P_{11}^t \quad (2.27)$$

$$P_{01}^{t+1} = P_{01}^t \cdot \frac{f(01)}{\bar{f}} \cdot [1 - P_c \cdot \frac{f(10)}{\bar{f}} \cdot P_{10}^t] + P_c \cdot \frac{f(00) \cdot f(11)}{\bar{f}^2} \cdot P_{00}^t \cdot P_{11}^t \quad (2.28)$$

$$P_{00}^{t+1} = P_{00}^t \cdot \frac{f(00)}{\bar{f}} \cdot [1 - P_c \cdot \frac{f(11)}{\bar{f}} \cdot P_{11}^t] + P_c \cdot \frac{f(01) \cdot f(10)}{\bar{f}^2} \cdot P_{01}^t \cdot P_{10}^t \quad (2.29)$$

其中 P_j^t 是模式在第 j 代中样本的比例, 变量 \bar{f} 是当前代中群体的平均适应度, 以式(2.30)表示:

$$\bar{f} = P_{00}^t \cdot f(00) + P_{01}^t \cdot f(01) + P_{10}^t \cdot f(10) + P_{11}^t \cdot f(11) \quad (2.30)$$

参数 P_c 是交叉发生在模式的定义距内的概率:

$$P_c = P_c \cdot \delta(H)/(l-1) \quad (2.31)$$

其中, P_c 为发生交叉的概率。

式(2.26)~(2.29)描述了模式 II 在子代中的样本比例, 在给定初始样本后, 就可以逐代跟踪各模式样本比例的变化。显然, 对于该

问题,遗传算法收敛的一个必要条件是全局最优模式的样本比例在足够多代后趋近于 1,即

$$\lim_{t \rightarrow \infty} P_{11}^t = 1 \quad (2.32)$$

图 2.3 示出了类型 I 的一个实例的结果。可以看到,开始最优模式(模式 11)的样本数减少,然而,随着模式 10 和 00 逐渐减少,模式 01 和 11 的样本逐渐占据主要地位,而后模式 01 逐渐减少,而模式 11 增加,最终遗传算法收敛到全局最优模式 11。一般地,如果 4 个模式的初始样本数非 0,则任何类型 I 骗问题都能收敛到全局最优解。这个结果多少令人感到惊讶,因为原本是设计一个使得遗传算法偏离全局最优解的问题。总之,动态分析表明类型 I 最小骗问题并非 GA-难的。

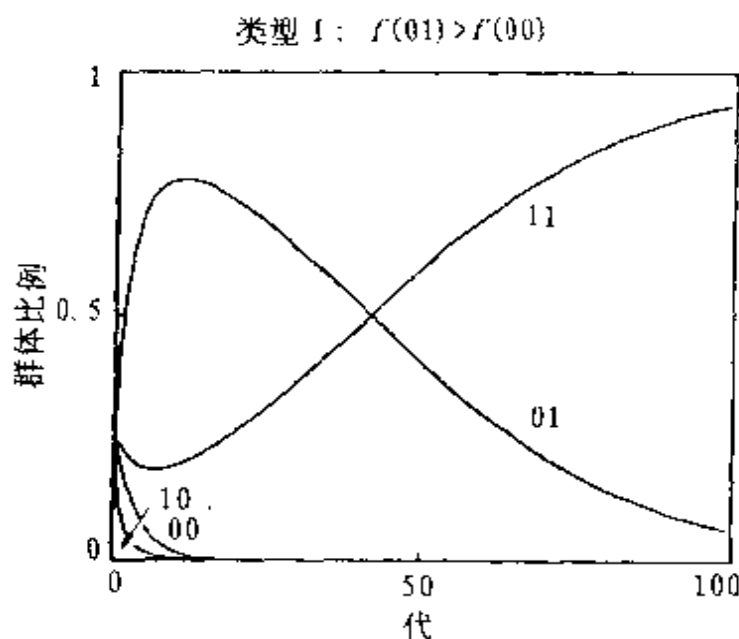


图 2.3 类型 I 最小骗问题数值解: $r=1.1, c=1.05, c'=0.0$

图 2.4 和图 2.5 示出了类型 II 最小骗问题的计算结果。从图 2.4 可以看出遗传算法收敛到了全局最优解。然而,并不是所有的类型 II 问题都能最终收敛到全局最优解。当给定模式 00 很大的初始样本数时,模式 11 可能被丢弃,而最终收敛到次最优解,图 2.5 表示这

种情况。针对类型Ⅱ问题 Goldberg^[9]给出了其收敛的充分条件。同样,多少令人感到惊讶的是,对于大多数初始条件,类型Ⅱ都收敛到

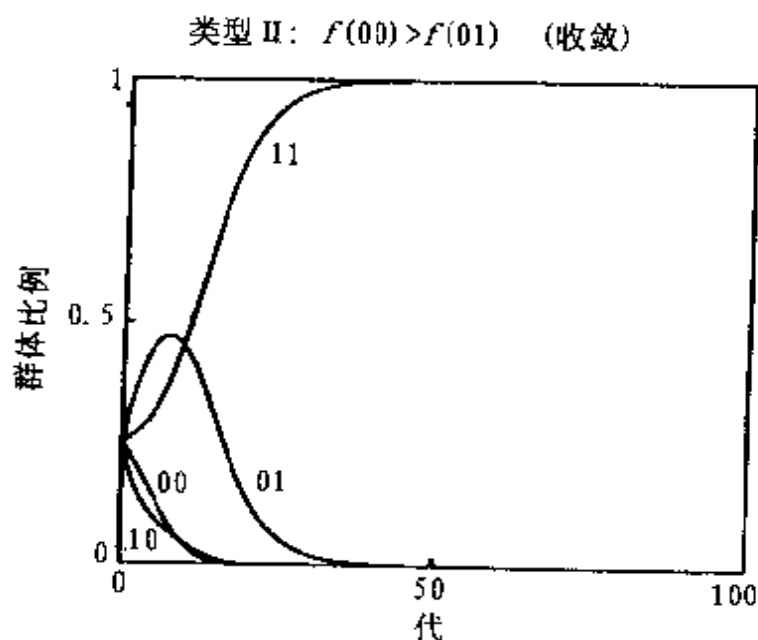


图 2.4 类型Ⅱ最小骗问题数值解(收敛): $r=1.1, c=0.9, c'=0.5$

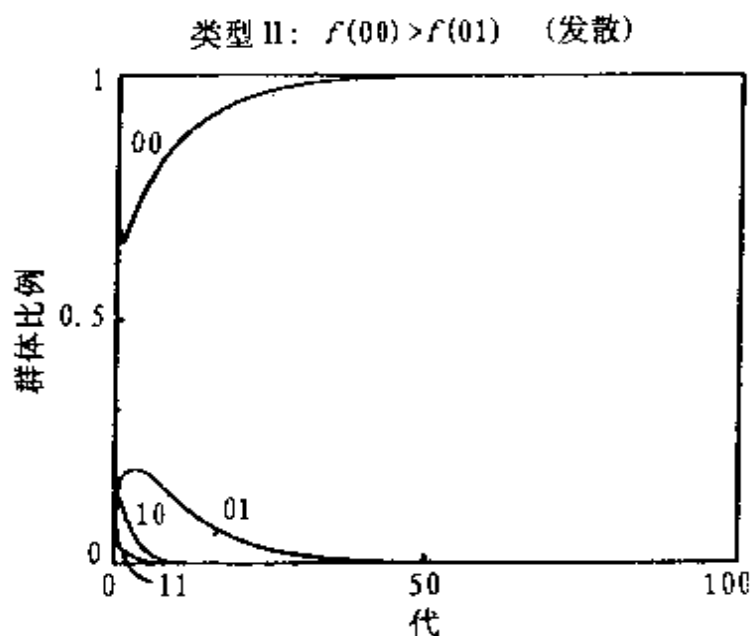


图 2.5 类型Ⅱ最小骗问题数值解(发散): $r=1.1, c=0.9, c'=0.5$

了全局最优解。最近的有关研究^[10,11]将上述模式分析推广到更高阶数的问题。其它有关的工作有助于我们定义所谓的 GA-难问题。当然,以上分析都是基于固定的编码方式(二进制编码)和简单的遗传操作(选择、交叉、变异)。采用复杂的遗传操作,比如增加逆转(inversion)操作,可能解决简单遗传算法难以解决的问题。

2.4 隐并行性

我们已经知道,一个串实际上隐含着多个模式,遗传算法实质上是模式的运算。对于一个长度为 l 的串,其中隐含着 2^l 个模式。那么,若群体规模为 n ,则其中隐含的模式个数介于 2^l 和 $n \cdot 2^l$ 之间。显然,由于交叉操作的作用并非所有的模式都能高概率地处理,因为一些定义距较长的模式将遭到破坏。这一节来讨论一下遗传算法中能以指数级增长的模式个数的下界。

在这方面,Holland 和 Goldberg^[12]指出了遗传算法有效处理的模式个数为 $O(n^3)$,这意味着,尽管遗传算法实际上只对 n 个串个体进行运算,但却隐含地处理了 $O(n^3)$ 个模式。Holland 命名这一性质为隐并行性(implicit parallelism)。以下通过一定的分析来推导这个结论。

考虑由长度为 l 的串所构成的规模为 n 的群体。我们只考虑那些生存概率大于 P_s 的模式(其中 P_s 为一常数),即在单点交叉和低概率变异情况下,那些丢失概率为 $\epsilon < 1 - P_s$ 的模式。因此,我们考虑那些定义距 $l_s < \epsilon(l-1) + 1$ 的模式。

不妨看一个例子。假设想计算定义距小于 $l_s = 5$ 的隐含在下面长度 l 为 10 的串中的模式:

1 0 1 1 1 0 0 0 1 0

首先,我们计算前 5 个位置的模式数,即

1 0 1 1 1 0 0 0 1 0

则第 5 位是固定的,即计算下列模式的个数:

% % % % 1 * * * * *

其中,“*”为无关符;而“%”既可以代表确定值(0 或 1),也可代表“*”。显然这样的模式数为 $2^l - 1 = 16$ 。为了计算整个串中的这类模式,我们将上面的底线向右移动一位,即

1 0 1 1 1 0 0 0 1 0

一般地,对于长度为 l 的串,计算定义距小于 l_s 的模式数,需将上述底线移动 $l - l_s + 1$ 次,由此得出一个长度为 l 的串,定义距小于等于 l_s 的模式数为 $2^{l-1} \cdot (l - l_s + 1)$ 。对于群体数为 n ,则此类模式总数为 $n \cdot 2^{l-1} \cdot (l - l_s + 1)$ 。显然,这个结论在群体规模较大的情况下存在着重复计数的问题。为了更准确些,取群体数 $n = 2^{l_s/2}$,由此希望阶数高于或等于 $l_s/2$ 的模式最多重复计数一次。另一方面,考虑到模式数目的分布呈二项式分布,则阶数高于 $l_s/2$ 的模式与低于 $l_s/2$ 的模式的数目大致相等,各占一半。如果我们只考虑高阶的部分,则有关模式数的下界为

$$n_s \geq n \cdot (l - l_s + 1) \cdot 2^{l_s-2} \quad (2.33)$$

考虑到 $n = 2^{l_s/2}$,则有

$$n_s = (l - l_s + 1) \cdot n^3/4 \quad (2.34)$$

即有 $n_s = cn^3 = O(n^3)$,其中 c 为常数。由此得出结论:遗传算法有效处理的模式总数正比于群体数 n 的立方,即为 $O(n^3)$ 。

以上,我们简略地推导了遗传算法中有效处理的模式数,可以看到,尽管具有高阶、长定义距的模式在交叉算子和变异算子作用下遭到破坏,遗传算法在处理相对小数目的串时,仍然隐含地处理了大量的模式。

2.5 性能评估

遗传算法的实现涉及到前述的五个要素,而每个要素又对应不同的环境存在各种相应的设计策略和方法。不同的策略和方法决定了各自的遗传算法具有不同的性能或特征。因此,评估遗传算法的性

能对于研究和应用搜索遗传算法是十分重要的。

目前,遗传算法的评估指标大多采用适应度值。特别在没有具体要求的情况下,一般采用各代中最优个体的适应度值和群体的平均适应度值。以此为依据,DeJong 提出两个用于定量分析遗传算法的测度。这就是离线性能(off-line performance)测度和在线性能(on-line performance)测度。前者测量收敛性,后者测量动态性能。之所以使用离线和在线测度是为了强调两者在应用上的差别。一般来说,在离线应用时,优化问题的求解可以得到模拟,在一定的优化进程停止准则下,当前最好的解可以被保存和利用;在在线应用中,优化问题的求解必须通过真实的实验在线实现,其好处在于可以迅速地得到较好的优化结果。

1. 在线性能评估准则

定义 2.5 设 $X_e(s)$ 为环境 e 下策略 s 的在线性能, $f_e(t)$ 为时刻 t 或第 t 代中相应于环境 e 的目标函数或平均适应度函数,则 $X_e(s)$ 可以表示为

$$X_e(s) = \frac{1}{T} \sum_{t=1}^T f_e(t) \quad (2.35)$$

上式表明,在线性能可以用从第一代到当前代的优化进程的平均值来表示。比如,若在线性能用平均适应度描述,则通过简单计算从第一代到当前代的各代平均适应度值对世代数的平均值即可获得在线性能。此时式(2.35)中的 $f_e(t)$ 对应于各代的平均适应度。

2. 离线性能评估准则

定义 2.6 设 $X_e^*(s)$ 为环境 e 下策略 s 的离线性能,则有

$$X_e^*(s) = \frac{1}{T} \sum_{t=1}^T f_e^*(t) \quad (2.36)$$

其中, $f_e^*(t) = \text{best}\{f_e(1), f_e(2) \cdots f_e(t)\}$ 。式(2.36)表明,离线性能是特定时刻最佳性能的累积平均。具体地说,在进化过程中每进化一代就统计目前为止的各代中的最佳适应度或最佳平均适应度,并计算对进化代数的平均值。

在线和离线性能的评估准则一般仅应用于不同遗传算法性能的评估比较。

2.6 编 码

我们已知道,遗传算法主要是通过遗传操作对群体中具有某种结构形式的个体施加结构重组处理,从而不断地搜索出群体中个体间的结构相似性,形成并优化积木块以逐渐逼近最优解。由此可见,遗传算法不能直接处理问题空间的参数,必须把它们转换成遗传空间的由基因按一定结构组成的染色体或个体。这一转换操作就叫做编码,也可以称作(问题的)表示(representation)。

一般来讲,由于遗传算法的鲁棒性,它对编码的要求并不苛刻。实际上,大多数问题都可以采用前面所见过的基因呈一维排列的染色体表现形式,尤其是基于 $\{0,1\}$ 符号集的二值编码形式。然而,正如我们将在后面的论述中所见到的那样,编码的策略或方法对于遗传操作,尤其是对于交叉操作的功能有很大的影响。在很多情况下,编码形式也就决定了交叉操作,编码问题往往称作编码-交叉问题。因此,作为遗传算法流程中第一步的编码技术是遗传算法研究中需要认真考虑的课题。本节主要介绍编码评估准则以及各种编码技术。

2.6.1 编码问题

图 2.6 描述了问题空间和 GA 空间的对应关系。所谓问题空间是指由 GA 表现型个体(有效的候选解)集所组成的空间;所谓 GA 空间是指由基因型个体所组成的空间。在 GA 空间中,个体的迁移是由交叉和变异所决定的。个体因一次变异所能迁移的局部空间叫做变异近邻(mutation neighborhood);由两个个体进行一次交叉所能迁移的局部空间叫交叉近邻(crossover neighborhood)。在 GA 空间中,由于交叉近邻是由两个个体所决定的,所以它比变异近邻迁移要大些。这也反映了遗传算法由于其核心操作——交叉所具有的全

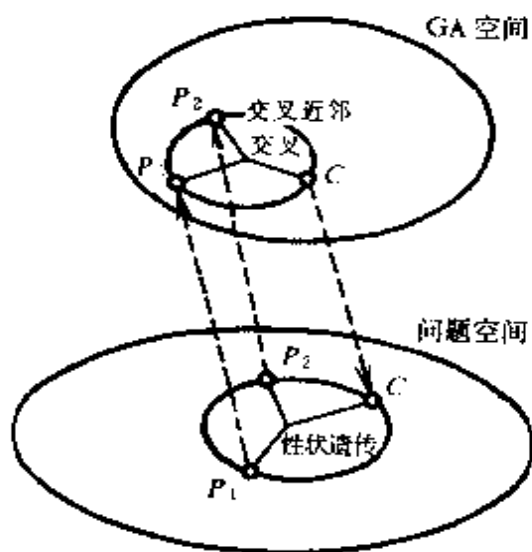


图 2.6 空间映射

进行逆映射,则在问题空间中可生成解个体 C 。显然,此个体已部分继承了父代 P_1 和 P_2 的性状。

需要指出的是,如果编码(包括译码)和交叉处理不当,在 GA 空间中因交叉而生成的个体逆映射到问题空间时,有可能成为无用解。这里称形成无用解的染色体为致死基因。同时,即使逆映射到问题空间的是有用解,但也可能是和双亲完全无缘的解。毫无疑问,我们不希望在问题空间中有这些不适当的解个体形成,这里,编码要发挥一定的作用^[14]。

需要再次强调的是,编码策略和交叉策略是互为依存的。恰当地设计编码和交叉策略或方法,对于充分发挥遗传算法的功能是十分重要的。对此,我们将在第三章深入讨论。

2.6.2 编码(译码)评估规范和编码原理

评估编码策略常采用以下 3 个规范^[2]:

完备性(completeness):问题空间中的所有点(候选解)都能作为 GA 空间中的点(染色体)表现。

健全性(soundness):GA 空间中的染色体能对应所有问题空间

局搜索性。

定义 2.7 由问题空间向 GA 空间的映射称作编码(encoding),而由 GA 空间向问题空间的映射称作译码(decoding)。

在图 2.6 中,从问题空间指向 GA 空间的虚线表示表现型向基因型的映射或转换,而由 GA 空间指向问题空间的虚线表示基因型向表现型的逆映射或逆转换。在 GA 空间中,对个体 P_1 和 P_2 实行交叉可生成个体 C 。对 C

中的候选解。

非冗余性(nonredundancy):染色体和候选解一一对应。

应该注意的是,严格满足上述规范的编码方法和提高遗传算法的效率并无关系。在有些场合,允许生成致死基因的编码,虽然这会导致冗余的搜索,但总的计算量可能反而减少,从而可以更有效地找出最优解。

上述的3个评估规范是独立于问题领域的普遍准则。因此,对于某个具体的应用领域而言,应该客观地比较和评估该问题领域中所用的编码(如交叉)的方法,由此得到更好的方法或策略。

上述的3个编码评估规范虽然带有普遍的意义,但是缺乏具体的指导思想,特别是满足这些规范的编码设计不一定能有效地提高遗传算法的搜索效率。相比之下,DeJong提出较为客观明确的编码评估准则,他称之为编码原理。由于其可操作性较强,常常又称之为编码规则。

DeJong的编码原理包括两条规则^[2]:

(1) 有意义积木块编码规则:所定编码应当易于生成与所求问题相关的短距和低阶的积木块。

(2) 最小字符集编码规则:所定编码应采用最小字符集以使问题得到自然的表示或描述。

规则(1)是基于模式定理和积木块假设。从应用的角度看,把握和应用问题空间相对应的积木块的编码结构是十分困难的。尽管如此,在编码设计中还是应该检查一下码串中相关位(bit)位置之间的距离。本章稍后将介绍一种对码串重新排序的方法。同时也将给出几种能在搜索优解的过程中同时搜索优质编码的技术和方法。

规则(2)提供了一种更为实用的编码原则。迄今为止,我们一直采用二值编码的设计,这不是偶然的,因为二值编码符合规则(2)的设计思想。让我们对此作更进一步的讨论。

前一章中,我们曾介绍用遗传算法求 $f(x)=x^2$ 函数的极值的实例,其中采用的是无符号二进制5位编码方法。对应于这种编码,

这里给出了另一种非二值编码。该编码是建立在一个由 32 个字符组成的字符集的基础上,其中包括{A—Z}26 个字符集和{1—6}个数字集。这两种编码都是针对整数集{0,31}而言,它们之间的部分对应关系如表 2.3 所示。显然,这里的二值编码是采用了最小的字符集{0,1},而非二值编码采用了一个较大的字符集{A—Z,1—6}。

表 2.3 **二值串群体和非二值串群体的比较**

| 二进制串 | X 值 | 非二值串 | 适应度 |
|-------|-----|------|-----|
| 01101 | 13 | N | 169 |
| 11000 | 24 | Y | 576 |
| 01000 | 8 | I | 64 |
| 10011 | 19 | T | 561 |

由表 2.3 可见,在二值编码的情况下,群体码串的相似性很容易找到;在非二值编码的情况下,由于表 2.3 的群体中仅有 4 个单一的码串,码串的相似性很难观察到。

为了能定量地说明问题,让我们对二值编码和非二值编码中可用的模式数作一比较。当然,这两种编码方法应该对相同数目的解进行编码,但是不同的字符集决定了它们需要不同的码串长度。为了保证空间中的解的数目相等,则应该使

$$2^l = K^{l'} \quad (2.37)$$

其中 l 为二进制编码的码串长度, l' 为非二值编码的码串长度, K 为非二值字符集中字符数目。在二值编码时所能得到模式数应该是 3^l (因为 l 个位置中的每一个均可为 0,1,*)。类似地,在非二值编码时的模式数应为 $(K+1)^{l'}$ 。显而易见,二值编码的每位信息可提供最多的模式数。由于这些相似性模板是遗传算法搜索的本质对象,所以,一个编码的设计应该能提供最多的模式,而采用最小字符集的编码原则为此提供了依据。

应该看到,上述的两个编码原理也仅仅是为编码设计提供了一

定的准则。在实际应用这些准则时仍然要针对具体问题,设计出具体有效的编码。以 TSP 问题为例,图 2.7 给出该问题(4 个城市 TSP 问题)的一个解以及相应的两种编码设计。

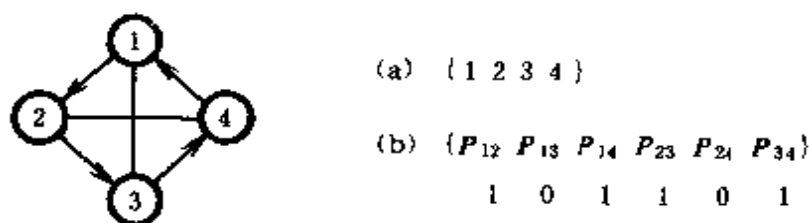


图 2.7 TSP 编码

图 2.7 中,编码(a)基于由 4 个城市编号按访问顺序组成的字符集,编码(b)是二值编码,其中,城市间路径与基因座对应,每个路径使用与否分别对应基因值 1 或 0。显然,编码(b)更符合前述的编码规则(2)。另一方面,就编码规则(1)而言,由于编码(a)只要考虑每个城市编号仅在码串中出现一次的条件就可以避免生成无意义的码串个体,即满足编码规则(1)。而编码(b)不易满足编码规则(1)。由此可见,前述的两个编码准则在具体应用中具有矛盾性。因此,真正有效的编码设计应该在本节介绍的评估规范和准则的基础上,认真考虑编码与问题约束条件的关系,编码与遗传操作尤其是和交叉操作的关系。

2.6.3 编码技术

下面,我们将介绍遗传算法的各种编码技术。由于编码设计往往不能与交叉设计分开,所以在重点介绍编码技术的同时须简单地介绍相应的交叉技术。而有关这些交叉的详细介绍请参阅本书以后相关章节。

1. 一维染色体编码

所谓一维染色体编码是指搜索空间的参数转换到遗传空间后,其相应的基因呈一维排列构成染色体。具体地说,在遗传空间中,用以表示个体的字符集中的要素构成了字符串。如前述的 4 个城市

TSP 问题中的某个可行解可编码成:

$$\{a, b, c, d\} \text{ 或 } \{1, 2, 3, 4\}$$

其中, a, b, c, d 为城市名, $1, 2, 3, 4$ 为相应的城市编号。

一维染色体编码中最常用的符号集是二值符号集 $\{0, 1\}$ 。基于此符号集的个体呈二值码串。比如, 前述的求函数 $f(x) = x^2 (x \in [0, 31])$ 极值问题中的一个可行解 ($X=13$) 被表示为 01101。

需要指出的是, 二值编码是目前遗传算法中常用的编码方法。它具有以下的特点:

- 1) 简单易行;
- 2) 符合最小字符集编码原则;
- 3) 便于用模式定理进行分析, 因为模式定理就是以二值编码为基础的。

但是, 二值编码具有一定的映射误差, 特别是它不能直接反映出所求问题的本身结构特征, 因此很难满足前述的生成有意义积木块编码原则。目前一维编码已有许多新的提案和方法。其中包括实数表示、格雷码表示和表表示等^[15]。对于这些, 本书有关章节将予以介绍。这里暂且让我们对二值编码再作更进一步讨论。

为不失一般性, 我们用大写字母表示某个二值字符串, 而用带下标的小写字母表示此个体的基因。比如 7 位字符串 $A = 0111000$ 可表示成: $A = a_1 a_2 a_3 a_4 a_5 a_6 a_7$ 。其中 a_i 表示一个基因即表示一个二值的特征或检测器, 而二值 0 或 1 又叫做 a_i 的基因值。有时候, 个体的基因可以不按序排列, 比如一个位串 A' 可表示为

$$A' = a_2 a_6 a_4 a_3 a_7 a_1 a_5$$

自第一章以来, 我们都假定所有的基因值都与它对应的基因座或基因在位串的位置有关。但在自然界, 生物染色体的基因是与其基因座无关的。模仿这种机理, 可得到一种重置编码技术。稍后, 我们将对此予以介绍。

2. 多参数映射编码

在优化问题求解中常常会碰到多参数优化问题。比如, 在一个如

图 2.8 所示典型的倒立摆问题中要学习优化的控制参数有如下 3 个：

\dot{X}_t : 小车的速度

θ_t : 摆倾斜角度

$\dot{\theta}_t$: 摆的角速度

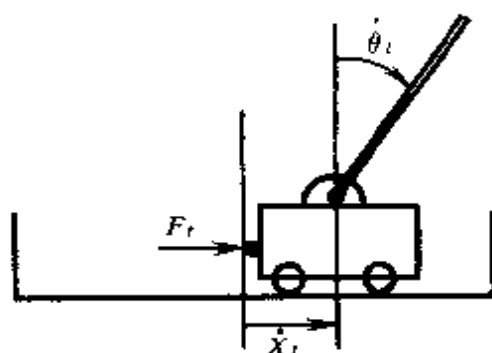


图 2.8 倒立摆

对此类问题的遗传算法常采用多参数编码。其基本思想是把每个参数先进行二值编码得到子串,再把这些子串连成一个完整的染色体。

假如子串的长度为 l , 则该子串译码后对应的无符号整数范围为 $[0, 2^l]$ 。又设某个参数的范围为 $[U_{\min}, U_{\max}]$ 。很明显, 两者之间存在一个映射关系, 而映射的编码精度 Π 为:

$$\Pi = \frac{U_{\max} - U_{\min}}{2^l - 1}$$

这种映射关系如下所示(子串长度为 4):

0 0 0 0 —————→ U_{\min}

⋮

1 1 1 1 —————→ U_{\max}

根据上述思想, 上述的 3 个参数优化的倒立摆控制问题的编码形式如下:

| | | | | |
|-------------|---|------------|---|------------------|
| 0 0 1 1 | ⋮ | 0 1 0 1 | ⋮ | 0 0 0 1 |
| \dot{X}_t | ⋮ | θ_t | ⋮ | $\dot{\theta}_t$ |

这是一个 12 位二进制码串, 其中包括 3 个子串分别对应于 \dot{X}_t , θ_t 和 $\dot{\theta}_t$ 。

一般来讲, 多参数映射编码中的每一个子串对应各自的编码参

数, 所以可有不同串长度和不同的 U_{\max} 和 U_{\min} 。

3. 离散化(discretization)

前述的多参数映射编码中曾提到了 $[0, 2^l]$ 到 $[U_{\min}, U_{\max}]$ 的映射操作。实际上, 这是一种离散化操作, 即将取值于 $[U_{\min}, U_{\max}]$ 范围的实参数(仅指定点小数)以长度为 l 的二进制码量化。在许多优化问题中, 尤其是优化控制问题中, 优化的不仅是一个控制参数而是一个连续的控制函数。在用遗传算法求解此类问题时, 需要把问题缩小成有限参数的形式后再对参数编码, 即进行离散化。

下面以一个例子来说明这种离散化操作。该例子是: 两地之间骑车所用最少时间。假定骑车所用用力 f 为时间 t 的函数, 且规定 $|f(t)| \leq f_{\max}$ 。这显然是一个求连续函数优化控制问题。此时, 骑车用力的分配可用图 2.9 所示的连续函数所描述。一般而言, 对于连续函数的离散化技术已很成熟。在这个例子中, 我们采用一定的时间间隔来离散化连续函数 $f(t)$, 然后再用阶跃函数、线性插值或三次抽样函数来逼近 $f(t)$ 。图 2.9 中采用了线性插值逼近的方法。

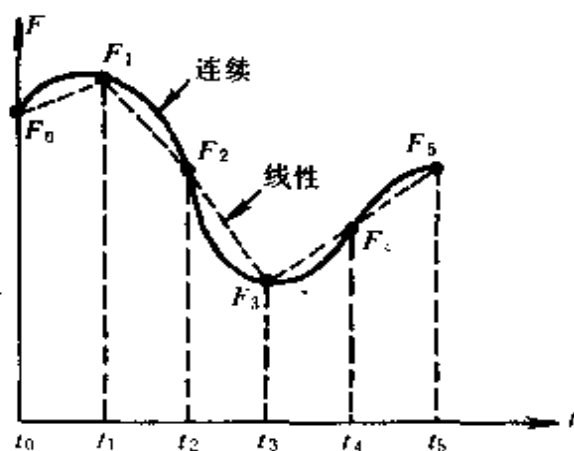


图 2.9 用力离散化

4. 可变染色体长度编码

在自然界生物进化过程中, 越是高等生物其染色体越长。换句话说, 生物进化过程中, 染色体的长度不是固定不变的。Goldberg 等提出的一种称为 MessyGA(mGA) 的遗传算法的编码方法就融进了这种机制。

mGA 有别于标准 GA(即 SGA)。它可以较好地克服 SGA 对于非线性问题(骗问题)处理的弱点。它主要有以下几个特点:

- 1) 染色体长度可变;

- 2) 允许过剩指定(overspecify)和缺省指定(underspecify);
- 3) 基于切断(cut)和拼接(splice)操作的交叉处理;
- 4) 分为二阶段的处理,即原始(Primordial)阶段和并立或对接(Juxtapositional)阶段。

这里我们仅介绍 mGA 的编码方法。关于 mGA 算法请参阅文献 [16]。

| | |
|------|-------------------------|
| 正确指定 | { (a1) (b4) (c3) } |
| 过剩指定 | { (a1) (b4) (c3) (a3) } |
| 缺省指定 | { (a1) (c3) } |

图 2.10 mGA 编码

图 2.10 给出了遗传算法的 mGA 的编码描述。在 mGA 中,个体仍然是由基因座以及相应的基因组成,但是基因的意义或特征和基因座无关,而是由成对的符号所决定。所以,在 mGA 中,个体被表示为表的形式,以图 2.10 中的个体 {(a1)(b4)(c3)} 为例(它也可看作是三位长的串),该表示可解释为 a 值为 1, b 值和 c 值分别为 4 和 3。

与 SGA 不同,mGA 不一定要把所有的信息都用基因表现出来,因此染色体的长度或表示个体的表的长度是可变的。设表示的基本字符集为 $\{a, b, c\}$, 染色体长度取 3, 则正确的个体表示或编码对应图 2.10 中的正确指定形式。染色体长度为 4 或为 2 的表示对应图 2.10 中的过剩指定和缺省指定形式。这 3 个染色体的长度分别为 3、4 和 2。

需指出的是,mGA 在做适应度评估计算时需要把以表形式表示的个体转换成规定长度的字符串的形式。此时对于过剩指定的表示,要把多余的指定除去;对缺损指定的表示,要用竞争模板来补足缺省。因此在 mGA 中,经适应度评估后,集团中染色体的长度都相等。但经过其特有的交叉操作后,新的群体中的染色体长度又可能不等长了。交叉操作如图 2.11 所示。此例中,两个亲个体的长为 9,

但经过切断和拼接操作后形成染色体长度为 6 和 12 的两个子代。

| | | | | |
|------|-------------------|---|------|-------------------------|
| 父代 1 | 1 1 1 1 1 1 1 1 1 | → | 子代 1 | 1 1 1 0 0 0 |
| 父代 2 | 0 0 0 0 0 0 0 0 0 | → | 子代 2 | 0 0 0 0 0 0 1 1 1 1 1 1 |
| | 切断 | | | 拼接 |

图 2.11 切断和拼接操作

mGA 的可变长染色体编码在有些应用中,尤其是在求解非线性问题时表现了很好的效果。但是,前述的模式定理已不适用于 mGA,有必要作适当的修正。

5. 二维染色体编码

在许多应用场合,问题的解呈二维或多维的表示。此时,若采用一维染色体编码则很不方便,尤其是交叉操作很不直观。这种场合宜采用多维编码。下面以图像恢复为例进行说明。

图像恢复是指把一个被干扰的图像尽量恢复到它的原始面目。显然,图像恢复处理所求的解是一个图像。用遗传算法来实现此任务时,一个染色体就代表一个图像。若图像为二值图像,则染色体就可表示成二维的二值数组,第八章图 8.1 给出了一个 8×8 的二值图像编码,其中每个基因对应一个像素。

由于利用二维染色体编码,所以相应的交叉操作也不同于一般的交叉操作。这时,可采用如第八章图 8.4 所示的纵横交叉和窗口交叉的方法。

很明显,由于二维图像可以看作是一维像素在二维空间中的按序排列,上述的交叉操作的交叉位置对于两个父体而言是对称一致的,所以模式定理在这种情况下依然成立。值得提出的是,由于图像的数据量极大,用遗传算法进行图像恢复时计算量很大。以 32×32 的二值图像为例,此问题相当于在 $32 \times 32 = 1024$ 维的空间中的 $2^{1024} = 10^{308}$ 个格子点中寻找满足目标函数的 1024 (32×32) 个格子点的问题。而且图像恢复属于不良设定问题,所以解不是唯一的。从这两点看,遗传算法对此问题只能得到局部解,需结合其它方法予以改进。关于这些请参阅本书第八章的内容介绍。

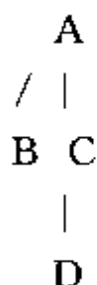
6. 树结构编码

前述的编码方法有若干缺点:1)个体的表示很难反映求解问题的本身结构或者层次;2)需要特定的编码和译码技术,把问题的解表示成合适的形式;3)不易表示高层次知识及相应的学习系统。

在实际应用中,许多问题本身可以采用结构描述。比如,人工智能中知识获取和提炼中采用的语义网络,神经网络中神经元连接状态图表示等。当然,此时也可以把图结构编码为一维染色体来处理,但是图结构特性不能在适应度评估中充分体现出来。作为对策,可直接把问题的结构表示作为染色体来处理,从而省去编码和译码操作^[17]。下面介绍一种树结构直接表示方法^[22]。

(1) 树结构表示及树距离

树结构是图的一种特殊形式,可表示如下:



树结构可用 LISP 语言中的 S 式来描述。比如上述中的树可描述为

(A(B)
(C(D)))

也可简化为

(A B
(C D))

所以,在以下的论述中,树结构就等效于 LISP 的 S 式。

为了形式地处理树结构,要引入树距离概念。这里先介绍对树进行的 4 个操作:

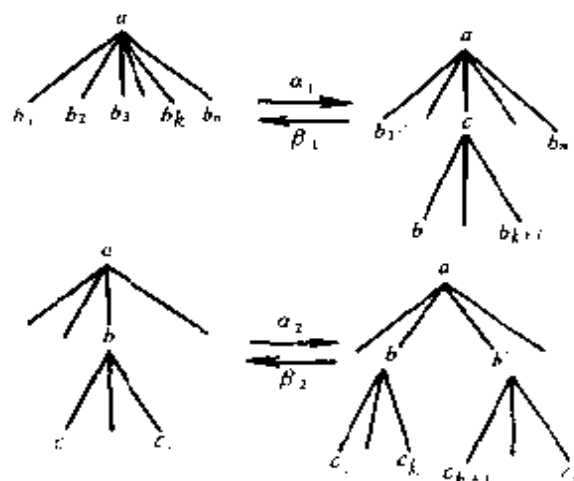


图 2.12 α - β 操作

- α_1 父子分割
 β_1 父子合并
 α_2 兄弟分割
 β_2 兄弟合并

图 2.12 给出了这些操作的例子。 α_1 操作把从 b_i 到 b_{k+i} 的部分树接到新的节点 c 下。

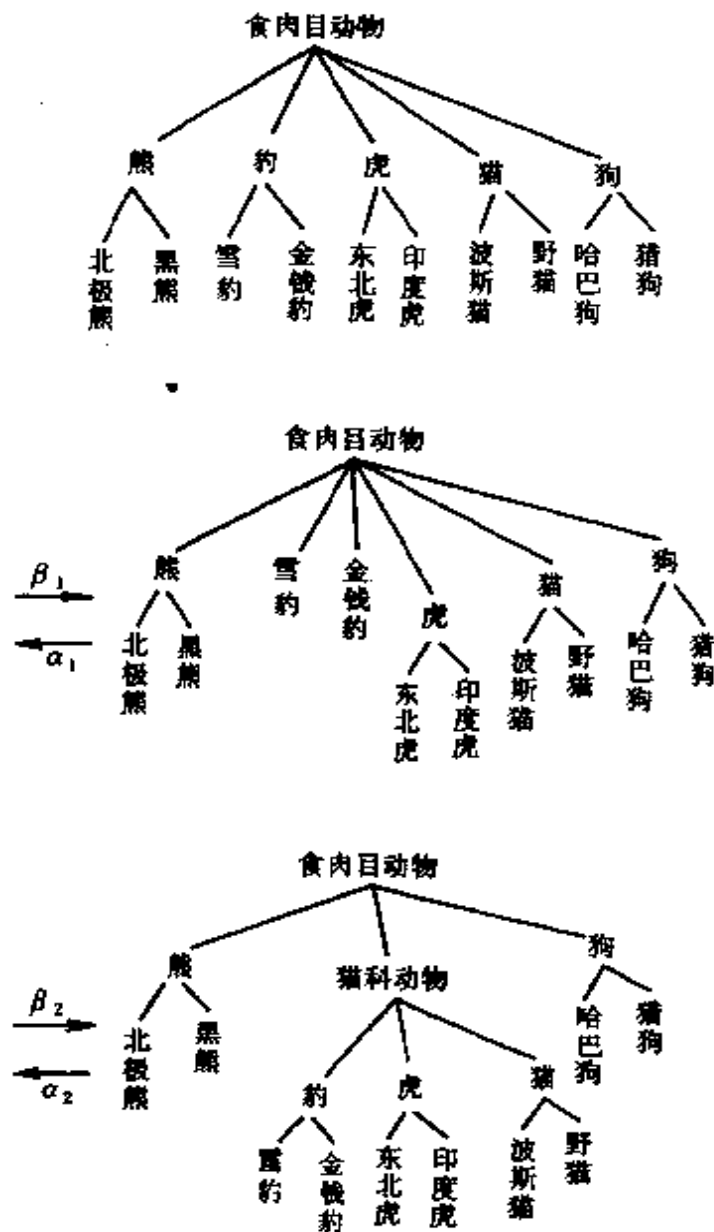


图 2.13 概念生成例

β_1 是 α_1 的逆操作。 α_2 操作是把以 b 为根的部分树分割为以 b 和 b' 为根的两个部分树。 β_2 是 α_2 的逆操作。为了进一步理解,我们以图 2.13 的食肉动物的概念图来说明。其中, α_1 操作是从各种豹例中形成称做豹的中间概念。把若干中间概念拼合起来形成猫科动物的新概念是由 β_2 操作完成的。显然, α_1 和 β_1 分别对应中间概念的生成和删除操作, α_2 和 β_2 分别对应通过事例合并形成概念或消除概念的操作。

树 T_1 和 T_2 的距离相应定义如下:

$$d(T_1, T_2) := \min \{ \#(M) \mid M \in \{ \alpha_1, \alpha_2, \beta_1, \beta_2 \}^* \wedge M(T_1) = T_2 \} \quad (2.38)$$

式中, $\#(M)$ 表示系列 M 的长度。 $M(T)$ 表示由 M 的算子对 T 进行变换所得到的树。 $d(T_1, T_2)$ 是把树 T_1 变换为 T_2 的最小系列的长

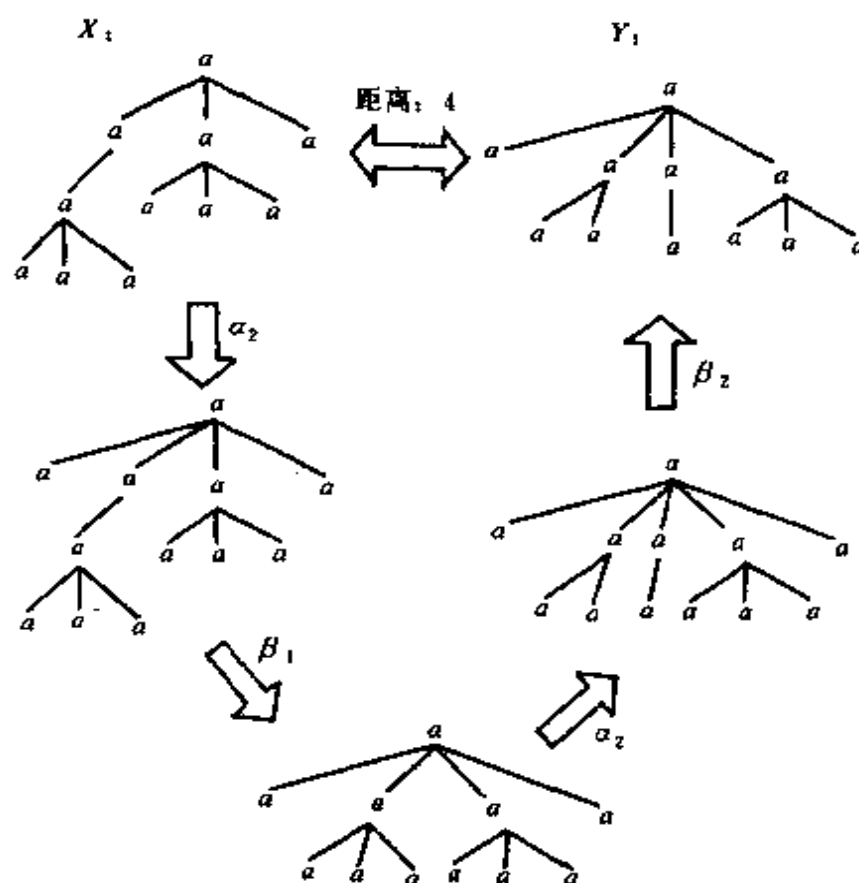


图 2.14 基于 α - β 算子的变换例

度。 d 满足距离公理, 即 $d(T, T) = 0, d(T_1, T_2) = d(T_2, T_1), d(T_1, T_2) + d(T_2, T_3) \geq d(T_1, T_3)$ 。此距离的计算复杂度是树节点数 n 的函数, 即 $O(n^3)$ 。图 2.14 表示了两棵树 X 和 Y 以及 α - β 算子作用的情况。

此变换可表示为

$$Y = \beta_2 \circ \alpha_2 \circ \beta_1 \circ \alpha_1(X)$$

可以证明此变换为最小系列变换, 所以, 树 X, Y 之间的距离为 4。

(2) 遗传算法算子

对于树结构的遗传操作如图 2.15 所示, 这是基本遗传操作的一

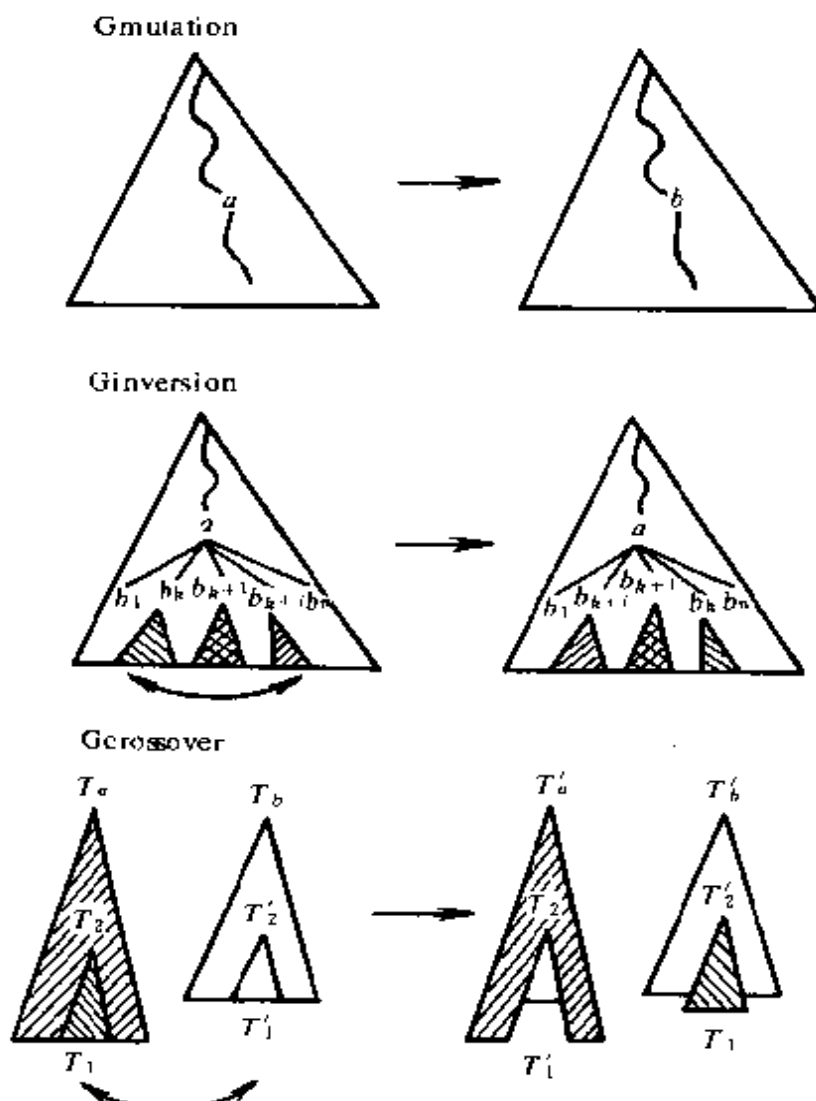


图 2.15 树结构遗传操作

种扩展。

这些算子可用于 LISP 的树形式表示,如图 2.16 所示。

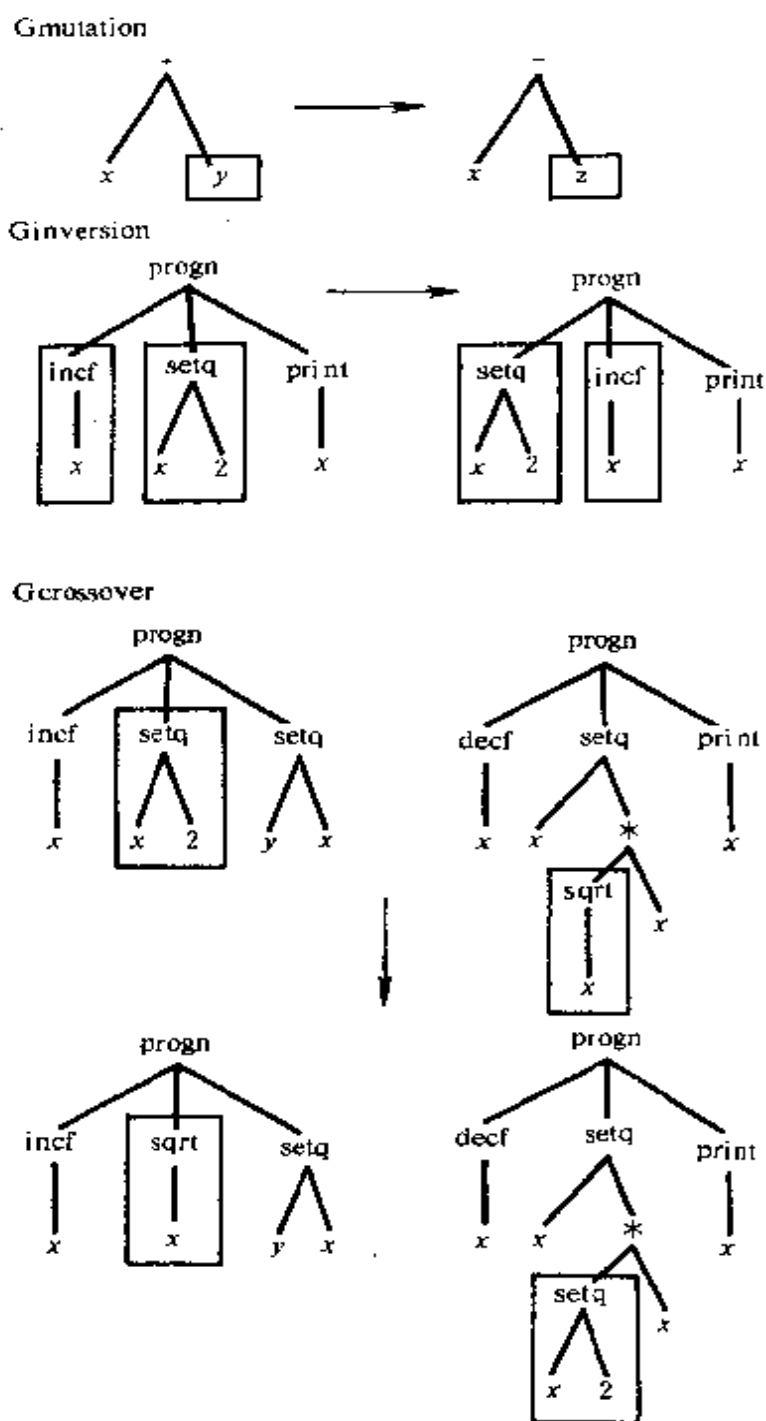


图 2.16 基于 S 式的变换

相应的 S 式描述如下所示,其中算子操作部位加下划线:

Gmutation $(+x\ y)$

↓

$(+x\ \underline{z})$

Ginversion $(\text{progn } (\text{incf } \underline{x}) (\text{setq } x\ 2) (\text{print } x))$

↓

$(\text{progn } (\underline{\text{setq } x\ 2}) (\text{incf } x) (\text{print } x))$

Gcrossover $(\text{progn}(\text{incf } x) (\underline{\text{setq } x\ 2}) (\text{setq } y\ x))$

$(\text{progn}(\text{decf } x) (\text{setq } x\ (*\ (\underline{\text{sqrt } x})x)) (\text{print } x))$

↓

$(\text{progn}(\text{incf } x) (\underline{\text{sqrt } x}) (\text{setq } y\ x))$

$(\text{progn}(\text{decf } x) (\text{setq } x\ (*\ (\underline{\text{setq } x\ 2})x)) (\text{print } x))$

这 3 个算子也可用前述的 α - β 来表示。特别是关于 Gcrossover, 下列关系成立:

$$d(T_a, T'_a) = d(T_b, T'_b) = d(T_1, T'_1)$$

$$d(T_a, T_b) = d(T'_a, T'_b) = d(T_1, T_2)$$

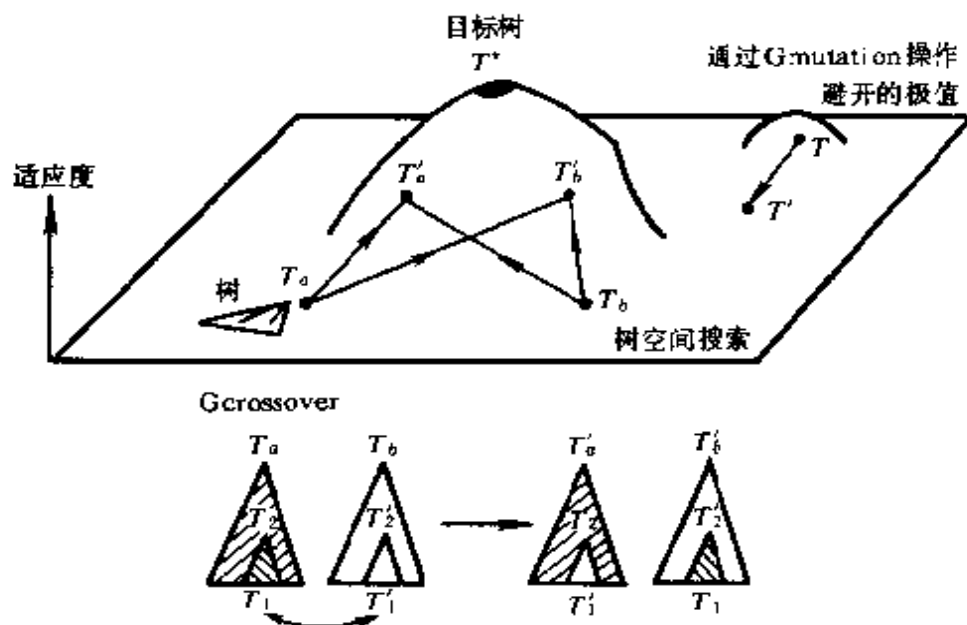


图 2.17 空间搜索

通过 Gcrossover 可搜索全部空间,如图 2.17 所示。

图 2.17 中, T_a 、 T_b 为 Gcrossover 的操作对象, T_a' 、 T_b' 是操作结果, T_1 、 T_1' 是 T_a 、 T_b 的 Gcrossover 具体操作部位,它们分别是 T_a 和 T_b 的部分树结构, T_2 、 T_2' 是 T_a 、 T_b 的残留部分。

基于上述结构表示的遗传操作的完整遗传算法框架与标准的遗传算法十分类似,但也有若干变动之处,关心此内容的读者请参阅文献[22]。

2.7 群体设定

遗传操作是对众多个体同时进行的。这众多的个体组成了群体。在遗传算法处理流程中,继编码设计后的任务是初始群体的设定,并以此为起点一代代进化直到按某种进化停止准则终止进化过程,由此得到最后一代(或群体)。关键问题是,群体规模,即群体中包括的个体数目如何设定?其中有两个需考虑的因素:1)初始群体的设定;2)进化过程中各代(或群体)的规模如何维持?无论怎样,作为遗传算法的控制参数之一,它和交叉概率、变异概率等参数一样,对于遗传算法效能的发挥是有影响的。

2.7.1 初始群体设定

模拟退火算法中,温度无限高的状态能量分布是通过随机状态的选择来设定的。同样,遗传算法中初始群体中的个体也是随机产生的。一般来讲,初始群体的设定可采取如下的策略:

(1) 根据问题固有知识,设法把握最优解所占空间在整个问题空间中的分布范围,然后,在此分布范围内设定初始群体。

(2) 先随机生成一定数目的个体,然后从中挑出最好的个体加到初始群体中。这种过程不断迭代,直到初始群体中个体数达到了预先确定的规模。

关于初始群体的具体设定事例请读者参阅本书第三章的介绍。

2.7.2 群体多样性

群体规模的确定受遗传操作中选择操作的影响很大。模式定理告诉我们,若群体规模为 M ,则遗传操作可从这 M 个个体中生成和检测 M^3 个模式,并在此基础上不断形成和优化积木块,直到找到最优解。显然 M 越大,遗传操作所处理的模式就越多,生成有意义的积木块并逐渐进化为最优解的机会就越高。换句话说,群体规模越大,群体中个体的多样性越高,算法陷入局部解的危险就越小。所以,从考虑群体多样性出发,群体规模应较大。但是,群体规模太大会带来若干弊病:一是从计算效率着眼,群体越大,其适应度评估次数增加,所以计算量也增加,从而影响算法效能;二是群体中个体生存下来概率(即后面一节将介绍的选择概率)大多采用和适应度成比例的方法,当群体中个体非常多时,少量适应度很高的个体会被选择而生存下来,但大多数个体却被淘汰^[18],这会影响配对库的形成,从而影响交叉操作。另一方面,群体规模太小,会使遗传算法的搜索空间中分布范围有限,因而搜索有可能停止在未成熟阶段,引起未成熟收敛(premature convergence)现象。显然,要避免未成熟收敛现象,必须保持群体的多样性,即群体规模不能太小。

我们知道,个体的多样性尺度是模式(schema)。对某字符集 G 而言,若个体的长度为 L ,则一个个体包含有 $(|G|+1)^L$ 个不同的模式。假定群体中有 M 个个体,若至少能匹配一个个体的模式数 $S(M, L, |G|, d)$ 越多,则群体个体的多样性就越丰富。这里, d 是表示 M 个个体中的个体重复度。若新的个体完全是随机生成,则 S 可用下式表示:

$$\begin{aligned}
 S(M, L, |G|, d) = & (|G| + 1)^L - \sum_{i=0}^L L^i |G|^i \\
 & \cdot \sum_{j=M-d+1}^M M^j \left(1 - \left(\frac{1}{|G|}\right)^i\right)^j \\
 & \cdot \left(\frac{1}{|G|}\right)^{i \cdot (M-j)}
 \end{aligned} \tag{2.39}$$

设 $|G|=2$ 且 $d=1$ (允许重复) 则有

$$\begin{aligned} S(M, L, 2, 0) &= 3^L - \sum_{i=0}^L L^{C_i} 2^i \left(1 - \left(\frac{1}{2}\right)^i\right)^M \\ &= 3^L - \sum_{j=0}^M M^{C_j} (-1)^j \cdot \left(1 + \left(\frac{1}{2}\right)^{j-1}\right)^L \\ &\approx 3^L - 3^L + M2^L = M2^L \end{aligned} \quad (2.40)$$

根据 schema 定理知 $S(M, L, 2, 0) = M^3$, 由此得 $M = 2^{L/2}$

这表明, 在二进制编码的前提下, 为了满足隐并行性, 群体个体数只要设定为 $2^{L/2}$ 即可。这个数比较大, 所以在实际应用中群体个体数的取值范围一般为几十~几百。

在遗传进化过程中, 群体的规模未必维持在相同的规模, 但一般情况下都维持相同规模。DeJong 在函数优化的实验中, 提出了一个描述群体之间关系的参数 G , 称为代沟 (generation gap)。通过 G 的引用, 可形成重叠群体:

$$G = \begin{cases} 1 & \text{非重叠群体} \\ 0 < G < 1 & \text{重叠群体} \end{cases}$$

在重叠群体模式下, 每次从群体中仅挑选 $n \cdot G$ 个个体参与遗传操作, 而其它个体直接保存到下一代中。

2.8 适应度函数

遗传算法在进化搜索中基本上不用外部信息, 仅用目标函数即适应度函数为依据。遗传算法的目标函数不受连续可微的约束且定义域可以为任意集合。对目标函数的唯一要求是, 针对输入可计算出能加以比较的非负结果。这一特点使得遗传算法应用范围很广。

在具体应用中, 适应度函数的设计要结合求解问题本身的要求而定。从本书后面的几章中, 读者可以充分理解针对各种具体应用的适应度函数设计方法。需要强调的是, 适应度函数评估是选择操作的依据, 适应度函数设计直接影响到遗传算法的性能。本节主要介

绍适应度函数设计的基本准则和要点，重点讨论适应度函数对遗传算法性能发挥的影响并给出相应的对策。

2.8.1 目标函数映射成适应度函数

在许多问题求解中，其目标是求取费用函数(代价函数) $g(x)$ 的最小值，而不是求效能函数或利润函数 $u(x)$ 的最大值。即使某一问题可自然地表示成求最大值形式，但也不能保证对于所有的 x ， $u(x)$ 都取非负值。由于遗传算法中，适应度函数要比较排序并在此基础上计算选择概率，所以适应度函数的值要取正值。由此可见，在不少场合，将目标函数映射成求最大值形式且函数值非负的适应度函数是必要的。

在通常搜索方法下，为了把一个最小化问题转化为最大化问题，只需要简单的把费用函数乘以 -1 即可，但对遗传算法而言，这种方法还不足以保证在各种情况下的非负值。对此，可采用以下的方法进行转换：

$$f(x) = \begin{cases} C_{\max} - g(x) & \text{当 } g(x) < C_{\max} \\ 0 & \text{其它情况} \end{cases}$$

显然存在多种方式来选择系数 C_{\max} 。 C_{\max} 可以是一个合适的输入值，也可采用迄今为止进化过程中 $g(x)$ 的最大值或当前群体中 $g(x)$ 的最大值。当然 C_{\max} 也可以是前 K 代中 $g(x)$ 的最大值。 C_{\max} 最好与群体无关。

当求解问题的目标函数采用利润函数形式时，为了保证其非负性，可用如下变换式：

$$f(x) = \begin{cases} u(x) + C_{\min} & \text{当 } u(x) + C_{\min} > 0 \\ 0 & \text{其他情况} \end{cases}$$

式中系数 C_{\min} 可以是合适的输入值，或是当前一代或前 K 代中的 $g(x)$ 的最小值，也可以是群体方差的函数。

2.8.2 适应度函数定标(scaling)

在自然界中,适应度和幸存再生的后代数是一个彼此相关的概念。大量的后代幸存是由于它们能适应环境,而其之所以能适应又是因为它们有大量的后代幸存。在自然界中,幸存是最终目标。为此,有必要,也有机会来调整群体中成员的竞争水平,以便获得适应度最高的个体。

应用遗传算法时尤其用它来处理小规模群体时常常会出现一些不利于优化的现象或结果。在遗传进化的初期,通常会出现一些超常的个体。若按照比例选择策略,这些异常个体有可能在群体中占据很大的比例,这是我们所不期望的现象,因为可导致未成熟收敛现象。显然,这些异常个体因竞争力太突出会控制选择过程,从而影响算法的全局优化性能。此外,在遗传进化过程中,虽然群体中个体多样性尚存在,但往往会出现群体的平均适应度已接近最佳个体适应度,这种情况下,个体间竞争力减弱,最佳个体和其它大多数个体在选择过程中有几乎相等的选择机会,从而使有目标的优化过程趋于无目标的随机漫游过程。显然,对于未成熟收敛现象,我们应设法降低某些异常个体的竞争力,这可以通过缩小相应的适应度函数值来实现。对于随机漫游现象,我们应设法提高个体间竞争力,这可以通过放大相应的适应度函数值来实现。这种对适应度的缩放调整称作适应度定标。自从 De Jong 开始引入适应度函数的定标问题以来,定标已成为保持进化过程中竞争水平的重要技术。目前,定标方式大致有以下几种。

(1) 线性定标(linear scaling)

设原适应度函数为 f , 定标后的适应度函数为 f' , 则线性定标可采用下式表示:

$$f' = af + b \quad (2.41)$$

式中,系数 a 和 b 可以有多种途径设定,但要满足两个条件:

- 1) 原适应度平均值 f 要等于定标后的适应度平均值。

2) 定标后适应度函数的最大值 f'_{\max} 要等于原适应函数平均值 f_{avg} 的所指定的倍数。即

$$f'_{\max} = C_{\text{mult}} \cdot f_{\text{avg}}$$

其中, C_{mult} 是为得到所期待的最优群体个体的复制数。实验表明, 对于一个典型的不太大的群体而言 ($n=50 \sim 100$), C_{mult} 可在 1.2~2.0 范围内取值。

条件 1) 的提出是为了保证在以后的选择处理中平均每个群体中的个体可贡献一个期待的子孙到下一代。条件 2) 的提出是为了控制原适应度最大的个体可贡献子孙的数目。

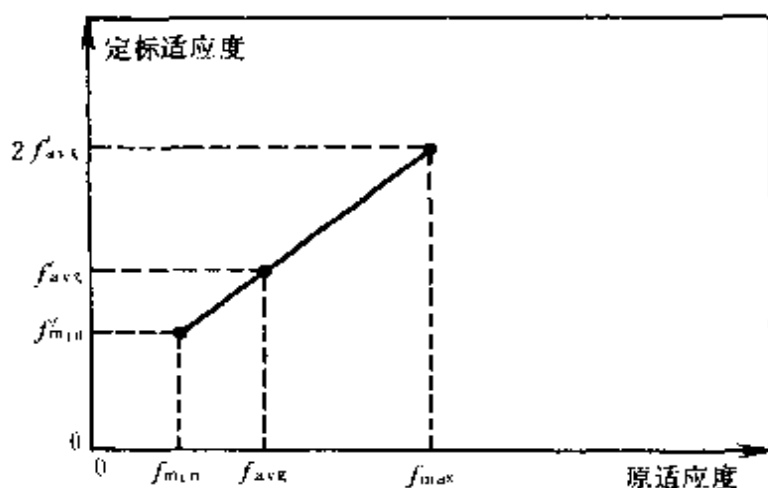


图 2.18 通常情况下的线性定标

必须提醒的是, 使用线性定标有可能出现负值适应度。究其原因, 这主要是在算法运行后期, C_{mult} 有可能对原适应度函数进行了过分的缩放。图 2.18 所示的情况属正常, 因为, 其中少数异常个体的适应度被缩小, 同时, 数量不多的个体适应度被扩大, 所以定标后的适应度未出现负值。但图 2.19 的情况就不同了。我们看到, 在一些个体(坏个体)的适应度值远小于群体的平均适应度值 f_{avg} 和最大适应度值 f_{\max} 且 f_{avg} 和 f_{\max} 又比较接近的情况下, 当采用线性定标欲把比较接近的 f_{\max} 和 f_{avg} 拉开时(为了提高群体个体的竞争力, 避免随机漫游现象), 原来低适应度值经定标后会变成负值。解决的方法很多, 当 C_{mult} 系数不能调整时, 我们可以简单地把原适应度最小值 f_{\min}

映射到定标后适应度最小值 f'_{\min} , 且使 $f'_{\min} = 0$ 。但此时仍需保持 $f'_{\text{avg}} = f'_{\text{avg}}$ 。

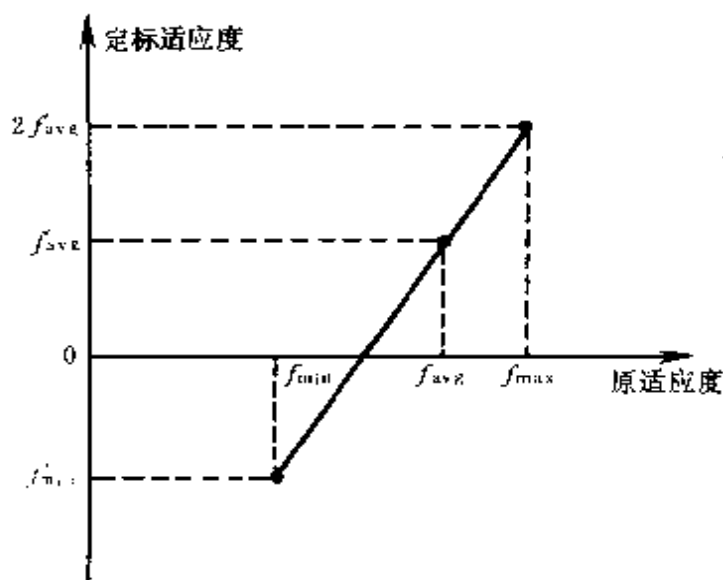


图 2.19 使负适应度值出现的线性定标

(2) σ 截断(sigma truncation)

σ 截断是使用上述线性定标前的一个预处理方法, 目的在于更有效地保证定标后的适应度值不出现负值。这是由 Forrest 提出的方法, 主要利用了群体标准方差信息来对适应度进行预处理。相应的表示式如下:

$$f' = f - (\bar{f} - c\sigma) \quad (2.42)$$

式中, 常数 c 要适当选择。

(3) 乘幂标

该定标方式可直接定义如下:

$$f' = f^K \quad (2.43)$$

式中, 幂指数 K 与求解问题有关, 而且在算法过程中可按需要修正。该定标方式由 Gillies 提出。他曾在机器视觉实验中采用了此方法, 当时, 他取 $K = 1.005$ 。

2.8.3 适应度函数的设计对遗传算法的影响

我们刚才已讨论了采用定标技术的适应度函数的设计可克服未成熟收敛和随机漫游现象,从而提高遗传算法的优化搜索性能。适应度函数的设计和遗传算法中的选择操作直接有关,所以它对遗传算法的影响还表现在其它方面。

1. 适应度函数影响遗传算法的迭代停止条件

严格地讲,遗传算法的迭代停止条件目前尚无定论。当适应度函数的最大值已知或者准最优解适应度的下限可以确定时,一般以发现满足最大值或准最优解作为遗传算法迭代停止条件。但是,在许多组合优化问题中,适应度最大值并不清楚,其本身就是搜索的对象,因此适应度下限很难确定。所以,在许多应用事例中,若发现群体个体的进化已趋于稳定状态,换句话说,若发现占群体一定比例的个体已完全是同一个体,则终止算法迭代。

2. 适应度函数与问题约束条件

遗传算法由于仅靠适应度来评估和引导搜索,所以求解问题所固有的约束条件不能明确地表示出来。在实际应用中,许多问题都是带约束条件的,像货郎担问题就是一个典型的约束组合优化问题。用遗传算法求解此类问题需要考虑一些对策。

按理说,我们可以采用一种十分自然的方法来考虑约束条件,即在进化过程中,迭代一次就设法检测一下新的个体是否违背了约束条件。如果没有违背,则作为有效个体,反之,作为无效个体被除去。这种方法对于弱约束问题求解还是有效的,但对于强约束问题求解效果不佳。这是因为在这种场合,寻找一个有效个体的难度不亚于寻找最优个体。

作为对策,可采取一种惩罚方法(penalty method)。该方法的基本思想是设法对个体违背约束条件的情况给予惩罚,并将此惩罚体现在适应度函数设计中。这样,一个约束优化问题就转换为一个附带考虑代价(cost)或惩罚(penalty)的非约束优化问题^[19]。

例如,一个原本是约束最小化问题可描述如下:

$$\text{最小化: } g(x)$$

$$\text{满足: } b_i(x) \geq 0 \quad i=1,2,\dots,n$$

其中 x 是一向量。

通过惩罚方法,上述问题可转化为非约束问题:

$$\text{最小化: } g(x) + r \cdot \sum_{i=1}^M \Phi[b_i(x)]$$

其中, Φ 为惩罚函数, r 为惩罚系数。

惩罚函数有许多确定方法。这里,我们对所有的违背约束条件的个体均按如下设定:

$$\Phi[b_i(x)] = b_i^2(x)$$

在一定的条件下,当惩罚系数 r 的取值接近无穷大时,非约束解可收敛到约束解。在实际应用中,遗传算法中 r 值通常对各类约束分别取值,这样可使对约束违背的惩罚份量将是适当的。把惩罚加到适应度函数中的思想是简单而直观。但是,惩罚函数值在约束边界处会发生急剧的变化,常常引起问题,需加以注意。用遗传算法求解约束问题的对策除了从适应度函数设计着手外,还可以在编码设计和遗传操作设计等方面采取一定的措施。前述的树结构编码方法就是一种直接把问题的结构特征即约束条件考虑进去的编码方法。有关处理约束条件的交叉操作设计请参照第三章有关内容。

2.9 遗传操作

遗传操作是模拟生物基因遗传的操作。在遗传算法中,通过编码组成初始群体后,遗传操作的任务就是对群体的个体按照它们对环境适应的程度(适应度评估)施加一定的操作,从而实现优胜劣汰的进化过程。从优化搜索的角度而言,遗传操作可使问题的解,一代又一代地优化,并逼近最优解。

遗传算法遗传操作包括以下三个基本遗传算子(genetic opera-

tor):1)选择(selection);2)交叉(crossover);3)变异(mutation)。这三个遗传算子有如下特点:

(1) 这三个遗传算子的操作都是在随机扰动情况下进行的。换句话说,遗传操作是随机化操作,因此,群体中个体向最优解迁移的规则是随机的。需要再次强调的是,这种随机化操作和传统的随机搜索方法是有区别的。遗传操作进行的是高效有向的搜索而不是如一般随机搜索方法所进行的无向搜索。

(2) 遗传操作的效果和上述三个遗传算子所取的操作概率,编码方法,群体大小,初始群体以及适应度函数的设定密切相关。对此,我们在前面的几节中已作了论述,后面还将继续对此进行讨论。

(3) 三个基本遗传算子的操作方法或操作策略随具体求解问题的不同而异。更具体地讲,是和个体的编码方式直接有关。由于目前二值编码仍是最常用的编码方法,所以,以下的对遗传算子的论述是以二值编码为基础的。相应于其它编码的遗传操作形式请参阅本书后面的章节。

2.9.1 选择算子

从群体中选择优胜的个体,淘汰劣质个体的操作叫选择。选择算子有时又称为再生算子(reproduction operator)。选择的目的是把优化的个体(或解)直接遗传到下一代或通过配对交叉产生新的个体再遗传到下一代。选择操作是建立在群体中个体的适应度评估基础上的,目前常用的选择算子有以下几种。

1. 适应度比例方法(fitness proportional model)

适应度比例方法是目前遗传算法中最基本也是最常用的选择方法。它也叫赌轮或蒙特卡罗(Monte Carlo)选择。在该方法中,各个个体的选择概率和其适应度值成比例。

设群体大小为 n , 其中个体 i 的适应度值为 f_i , 则 i 被选择的概率 P_{si} 为

$$P_{si} = f_i / \sum_{j=1}^M f_j \quad (2.44)$$

显然, 概率 P_{si} 反映了个体 i 的适应度在整个群体的个体适应度总和中所占的比例. 个体适应度越大, 其被选择的概率就越高, 反之亦然(见第一章第一节的表 1.2)。按式(2.44)计算出群体中各个个体的选择概率后, 就可以决定哪些个体被选出。该思想可用以下的算法描述:

```

procedure selection
begin
    i ← 0
    sum ← 0
    wheel-pos ← random * FSUM
    While sum ≤ wheel-pos    or i ≤ PSIZE do
        i ← i + 1
        sum ← sum + Fi
    endwhile
    str-no ← i
end

```

这里, FSUM 为群体中所有个体适应度和, F_i 为第 i 个个体的适应度, random 是在 $[0, 1]$ 区间内产生随机数的随机函数. wheel-pos 相当于赌轮环上的位置, str-no 是所选个体号。表 2.4 给出了采用适应度比例方法的适应度与选择概率的例子。这里适应度函数采用幂定标法($k=2$)。个体被选后, 可随机地组成交配对, 供后面的交叉操作(参见表 1.2)。

表 2.4 **适应度与选择概率**

| 个体编号 | 原适应度 | 定标后适应度 | 选择概率 |
|------|------|--------|------|
| 1 | 2.5 | 6.25 | 0.18 |
| 2 | 1.0 | 1.00 | 0.03 |
| 3 | 3.0 | 9.00 | 0.26 |

续表

| 个体编号 | 原适应度 | 定标后适应度 | 选择概率 |
|------|------|--------|------|
| 4 | 1.2 | 1.44 | 0.04 |
| 5 | 2.1 | 4.41 | 0.13 |
| 6 | 0.8 | 0.64 | 0.02 |
| 7 | 2.5 | 6.25 | 0.18 |
| 8 | 1.3 | 1.69 | 0.05 |
| 9 | 0.9 | 0.81 | 0.02 |
| 10 | 1.8 | 3.24 | 0.09 |

2. 最佳个体保存方法(elitist model)

该方法的思想是把群体中适应度最高的个体不进行配对交叉而直接复制到下一代中。此种选择操作又称复制(copy)。De Jong 对此方法作了如下定义:

定义 2.8 设到时刻 t (第 t 代) 时, 群体中 $a^*(t)$ 为最佳个体。又设 $A(t+1)$ 为新一代群体, 若 $A(t+1)$ 中不存在 $a^*(t)$, 则把 $a^*(t)$ 作为 $A(t+1)$ 中的第 $n+1$ 个个体 (其中, n 为群体大小)。

采用此选择方法的优点是, 进化过程中某一代的最优解可不被交叉和变异操作所破坏。但是, 这也隐含了一种危机, 即局部最优个体的遗传基因会急速增加而使进化有可能限于局部解。也就是说, 该方法的全局搜索能力差, 它更适合单峰性质的搜索空间搜索, 而不是多峰性质的空间搜索。所以此方法一般都与其他选择方法结合使用。

3. 期望值方法(expected value model)

在赌轮选择方法中, 当个体数不太多时, 依据产生的随机数有可能会出现不正确地反映个体适应度的选择, 即存在统计误差。也就是说, 适应度高的个体也有可能被淘汰 (当然, 适应度低的个体也有可能被选择), 为克服这种误差, 期望值方法用了如下思想。

(1) 计算群体中每个个体在下一代生存的期望数目：

$$M = f_i / \bar{f} = f_i / \sum f_i / n \quad (2.45)$$

(2) 若某个体被选中并要参与配对和交叉,则它在下一代中的生存的期望数目减去 0.5;若不参与配对和交叉,则该个体的生存期望数目减去 1。

(3) 在(2)的两种情况中,若一个个体的期望值小于零时,则该个体不参与选择。

以表 2.1 中的个体 1 为例,它的期望值若按式(2.45)计算,则为选择概率的 10 倍,即为 1.80,若它被选为配对交叉个体,则新的期望值为 1.3。

De Jong 曾对比了适应度比例选择法,最佳个体保存法和期望值法用于函数优化中的性能。对比实验表明,采用期望值法的遗传算法离线性和在线性都高于采用另外两种方法的遗传算法性能^[2]。

4. 排序选择方法(rank-based model)

所谓排序选择方法是指在计算每个个体的适应度后,根据适应度大小顺序对群体中个体排序,然后把事先设计好的概率表按序分配给个体,作为各自的选择概率。表 2.5 给出了排序选择方法中适应度与选择概率的关系的一个例子。由表可见,所有个体按适应度大小排序,而选择概率和适应度无直接关系而仅与序号有关。这种方法的不足之处在于选择概率和序号的关系需事先确定。此外,它和适应度比例方法一样都是一种基于概率的选择,所以仍有统计误差。

表 2.5 排序选择例

| 排序 | 个体 | 适应度 | 选择概率 |
|----|----|-----|------|
| 1 | 3 | 3.0 | 0.35 |
| 2 | 1 | 2.5 | 0.20 |
| 3 | 7 | 2.5 | 0.15 |
| 4 | 5 | 2.1 | 0.10 |

续表

| 排序 | 个体 | 适应度 | 选择概率 |
|----|----|-----|------|
| 5 | 10 | 1.8 | 0.06 |
| 6 | 8 | 1.3 | 0.05 |
| 7 | 4 | 1.2 | 0.04 |
| 8 | 2 | 1.0 | 0.03 |
| 9 | 9 | 0.9 | 0.02 |
| 10 | 6 | 0.8 | 0.00 |

5. 联赛选择方法(tournament selection model)

该方法十分简单,其操作思想是,从群体中任意选择一定数目的个体(称为联赛规模),其中适应度最高的个体保存到下一代。这一过程反复执行,直到保存到下一代的个体数达到预先设定的数目为止。联赛规模一般取 2。

6. 排挤方法(crowding model)

在自然界中,当一种群中的个体大量繁殖生存时,为争夺有限的生存资源,群体中个体之间的竞争压力必然加剧,因此,个体的寿命和出生率也因此而降低。基于这种机制,De Jong 提出了排挤选择方法。采用该方法时,新生成的子代将替代或排挤相似的旧父代个体。该方法可提高群体的多样性。

在采用覆盖群体模式的情况下(代沟 $G=0.1$),排挤方法可描述如下:

1) 设定参数 CF(crowding factory);

2) 从群体中随机地挑选 CF 个个体组成个体集(新的个体不包括在内);

3) 从这个体集中淘汰一个个体,该个体与新个体的海明距离最短。

以上介绍的是目前常用的几种选择方法。每种方法对于遗传算法的在线和离线性能的影响均各不相同。在具体使用时,应根据问

题求解特点采用较合适的方法或者是把它们结合使用。

2.9.2 交叉算子(crossover operator)

在自然界生物进化过程中起核心作用的是生物遗传基因的重组(加上变异)。同样,遗传算法中起核心作用的是遗传操作的交叉算子。正如第一章所介绍的那样,所谓交叉是指把两个父代个体的部分结构加以替换重组而生成新个体的操作。通过交叉,遗传算法的搜索能力得以飞跃提高。

1. 交叉算子设计要点

实现个体结构重组的交叉算子设计一般与所求解的具体问题有关,无论设计何种交叉算子均需考虑如下几点:

(1) 任何交叉算子需满足交叉算子的评估准则,即交叉算子需保证前一代中优秀个体的性状能在后一代的新个体中尽可能得到遗传和继承。

(2) 在第一节中我们已指出,编码设计和交叉设计是相互联系的。换句话说,要使设计出的交叉算子能满足上述评估准则,交叉算子设计和编码设计需协调操作。这也就是所谓的编码-交叉设计。

(3) 对于占主流地位的二值编码而言,各种交叉算子都包括两个基本内容:

1) 从由选择操作形成的配对库(mating pool)中,对个体随机配对并按预先设定的交叉概率来决定每对是否需要交叉操作;

2) 设定配对个体的交叉点(cross site),并对这些点前后的配对个体的部分结构(或基因)进行相互交换。

2. 基本的交叉算子

本节以字符串编码为前提介绍几种基本交叉方法。

(1) 一点交叉(one-point crossover)。

一点交叉又叫简单交叉。具体操作是:在个体串中随机设定一个交叉点,实行交叉时,该点前或后的两个个体的部分结构进行互换,并生成两个新个体。下面给出了一点交叉的例子。

| | | | | | | | |
|------|------|------|---|-----|-----|---------|--------|
| 配对个体 | 个体 A | 1001 | : | 111 | ——→ | 1001000 | 新个体 A' |
| | 个体 B | 0011 | : | 000 | ——→ | 0011111 | 新个体 B' |
| 交叉点 | | | | | | | |

其中,交叉点设置在第 4 和第 5 个基因座之间。交叉时,该交叉点后的两个个体的部分码串互相交换。这样,个体 A 的第一到第四个基因与个体 B 第五到第七个基因组成一个新的个体 A'。同样,可得到新个体 B'。交叉点是随机设定的,当染色体长为 n 时,可能有 $n-1$ 个交叉点设置,所以,一点交叉可能实现 $n-1$ 个不同的交叉结果。

(2) 二点交叉(two-point crossover)。

二点交叉的操作与一点交叉类似,只是设置 2 个交叉点(依然是随机设定)。一个二点交叉的例子表示如下:

| | | | | | | | | | |
|----------------|------|----|---|-----|---|----|-----|---------|--------|
| 配对个体 | 个体 A | 10 | : | 110 | : | 11 | ——→ | 1001011 | 新个体 A' |
| | 个体 B | 00 | : | 010 | : | 00 | ——→ | 0011000 | 新个体 B' |
| 交叉点 1 交叉点 2 | | | | | | | | | |

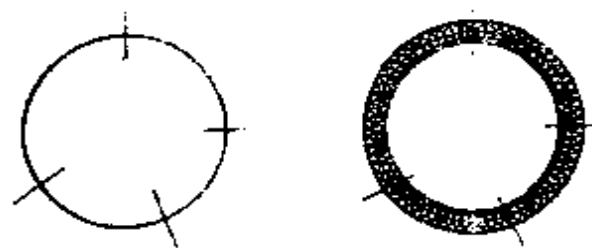
由此可见,2 个交叉点分别设定在第二基因座和第三基因座以及第五基因座和第六基因座之间。A,B 两个体在这两个交叉点之间的码串相互交换,分别生成新个体 A' 和 B'。对于二点交叉而言,若染色体长为 n ,则可能有 $(n-2)(n-3)$ 种交叉点的设置。

(3) 多点交叉(multi-point crossover)。

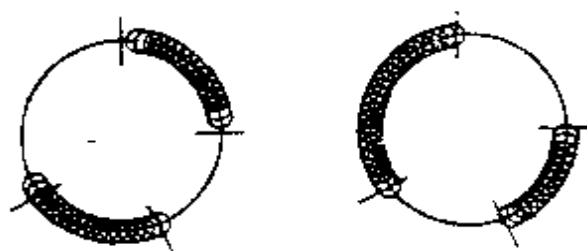
多点交叉是前述两种交叉的推广,有时又被称为广义交叉(generalized crossover)。若用参数 cp 来表示交叉点数,则当 $cp=1$ 时,广义交叉就等效于一点交叉。当 cp 为偶数时,根据交叉互换规律,染色体可以当作基因环来处理。比如,当 $cp=4$ 时,交叉情况如图 2.20 所示。当 cp 为奇数时,它可以等效为偶数点交叉。

此时,可假定在码串的头部(即基因座 0)有一个交叉点。图 2.21 给出了 $cp=3$ 时的交叉情况。

一般来讲,多点交叉较少采用,因为它影响遗传算法的在线和离

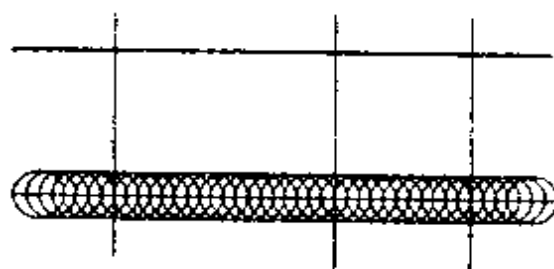


(a) 随机选择 4 个交叉点

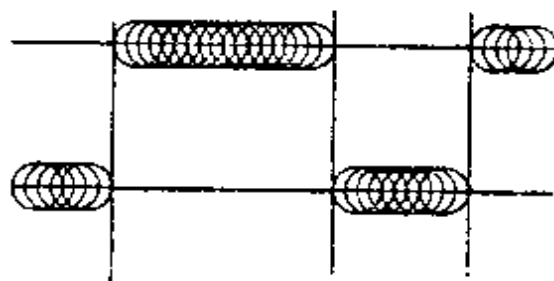


(b) 4 点交叉产生 2 个新环

图 2.20 偶数点交叉



(a) 随机选择 3 个交叉点



(b) 3 点交叉产生 2 个新环

图 2.21 奇数点交叉

线性能。事实上,若染色体长为 n ,则对多点交叉而言共有 $(\frac{n}{cp})$ 种交叉设置可供随机挑选。当 cp 较大时,每种交叉点设置被随机选中的可能性很小,因此能被保存的部分结构也就很少。这有些类似于随机搜索的行为。换一种讲法,多点交叉不能有效地保存重要的模式。

(4) 一致交叉(uniform crossover)。

所谓一致交叉是指通过设定屏蔽字(mask)来决定新个体的基因继承两个旧个体中哪个个体的对应基因。一致交叉的操作过程表示如下:当屏蔽字中的位为 0 时,新个体 A' 继承旧个体 A 中对应的基因,当屏蔽字位为 1 时,新个体 A' 继承旧个体 B 中对应的基因,由此生成一个完整的新个体 A' 。反之,可生成新个体 B' 。显然,一致交叉包括在多点交叉范围内。一个一致交叉的例子表示如下:

| | |
|----------|--------|
| 旧个体 A | 001111 |
| 旧个体 B | 111100 |
| <hr/> | |
| 屏蔽字 | 010101 |
| <hr/> | |
| 新个体 A' | 011110 |
| 新个体 B' | 101101 |

(5) 其它交叉方法

针对各个问题的不同,可以提出各种交叉方法。像第一节中我们曾提到的二维交叉, Messay GA 中的交叉,树结构交叉以及 TSP 问题中的部分匹配交叉(PMX),顺序交叉(OX)和周期交叉(CX)等。在第三章中将会介绍更多的具体交叉算子设计实例,请读者参阅。

3. 交叉与遗传算法收敛性

遗传算法的收敛性不仅取决于本章前面已讨论过的编码,初始群体,适应度函数,选择算子的设计,而且还取决于交叉算子和下面即将介绍的变异算子的设计。但是,遗传算法的收敛性主要决定于作为其核心操作的交叉算子。交叉算子收敛性是当今遗传算法研究

中最重要课题之一,目前仍无系统而全面的论述。但是,通过局部的理论分析,尤其是大量的卓有成效的实验,对于交叉收敛性已有若干共识。现大致规纳如下:

(1) 交叉算子并未提供收敛性保证。模式定理只是在考虑选择算子可维持模式而交叉和变异算子会破坏模式的基础上提供了通过不断的交叉,低阶、短距的模式可呈指数倍增长的论述,但它未能确认这呈指数增长的模式一定向最优解收敛^[2,20]。

(2) 交叉算子的有效工作范围以及其边界可通过骗定理或骗问题的分析而得到描述^[2,21]。

(3) 虽然交叉缺乏收敛保证,但这并不影响遗传算法的实用性。有时,许多收敛算法由于其收敛性却牺牲了它们的全局性和灵活性。因此,不少对于收敛算法是很难求解的问题,对于基于交叉的遗传算法而言反而是简单可解的了。

(4) 排除变异仅有交叉的遗传算法可等效为有限吸收的 Markov 过程^[23]。在适当的选择策略下,通过交叉可实现向最优解收敛。关于这一点,本章最后一节将进行讨论。

2.9.3 变异算子(mutation operator)

变异算子的基本内容是对群体中的个体串的某些基因座上的基因值作变动。就基于字符集{0,1}的二值码串而言,变异操作就是把某些基因座上的基因值取反,即 $1 \rightarrow 0$ 或 $0 \rightarrow 1$ 。一般来说,变异算子操作的基本步骤如下:

(1) 在群体中所有个体的码串范围内随机地确定基因座。

(2) 以事先设定的变异概率 P_m 来对这些基因座的基因值进行变异。

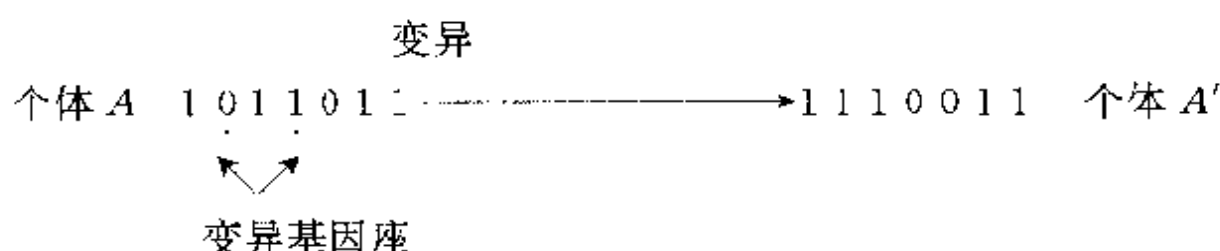
遗传算法导入变异的目的是有两个:一是使遗传算法具有局部的随机搜索能力。当遗传算法通过交叉算子已接近最优解邻域时,利用变异算子的这种局部随机搜索能力可以加速向最优解收敛。显然,此种情况下的变异概率应取较小值,否则接近最优解的积木块会因

变异而遭到破坏。二是使遗传算法可维持群体多样性,以防止出现未成熟收敛现象。此时收敛概率应取较大值。

遗传算法中,交叉算子因其全局搜索能力而作为主要算子,变异算子因其局部搜索能力而作为辅助算子。遗传算法通过交叉和变异这一对相互配合又相互竞争的操作而使其具备兼顾全局和局部的均衡搜索能力。所谓相互配合,是指当群体在进化中陷于搜索空间中某个超平面而仅靠交叉不能摆脱时,通过变异操作可有助于这种摆脱。所谓相互竞争,是指当通过交叉已形成所期望的积木块时,变异操作有可能破坏这些积木块。因此,如何有效地配合使用交叉和变异操作,是目前遗传算法的一个重要研究内容。

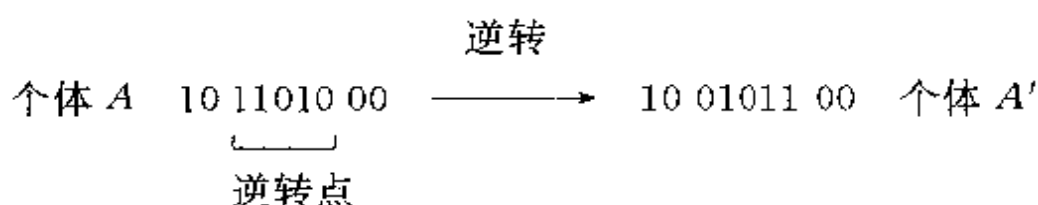
1. 基本变异算子

基本变异算子是指对群体中的个体码串随机挑选一个或多个基因座并对这些基因座的基因值做变动(以变异概率 P_m 做变动), $\{0, 1\}$ 二值码串中的基本变异操作如下:



2. 逆转算子(inversion operator)

逆转算子是变异算子的一种特殊形式。它的基本操作内容是,在个体码串中随机挑选二个逆转点,然后将两个逆转点间的基因值以逆转概率 P_i 逆向排序。 $\{0, 1\}$ 二值码串的逆转操作如下:



由此可见,通过逆转操作,个体中从基因座 3 到基因座 7 之间的基因排列得到逆转,即从 11010 序列变成了 01011 序列。这一逆转操作

可以等效为一种变异操作,但是逆转操作的真正目的并不在变异(否则仅用变异操作就行了)而在实现一种重新排序操作(reordering operator)^[2]。

所谓重新排序是指对个体中基因排列所进行重新组合,但并不影响该个体的特征。在自然界生物的基因重组中就有这种重新排序的机制。对遗传算法而言,采用这种重新排序的目的是为了提高积木块的繁殖率。实际上,在用遗传算法求解某些问题时,群体中的有些个体的基因排序常常会出现这样的情况,即对形成积木块有用的某些基因分离较远,此时采用一般的交叉会破坏相应的积木块的生成。因此,有必要对这些基因进行重新排序但又不损整个个体的特征(即适应度值)。

为了用逆转算子来实现这种重新排序功能必须把基因值的意义与其基因座位置独立开来,以保证经过重新排序的个体适应度不变。这里可采用如下所示的个体扩展表示法:

| | | | | |
|------|---|-------|-----------------|-------|
| 个体 A | 1 2 3 4 5 6 7 8 | 逆转点 | 1 2 7 6 5 4 3 8 | 个体 A' |
| | 1 0 1 1 0 1 0 0 | ————→ | 1 0 0 1 0 1 1 0 | |
| | <div style="text-align: center;">└──────────┘</div> | | | |
| | 逆转点 | | | |

此时,每个基因都用整数 1--8 命名或编号。这些命名或编号记述了各个基因与基因座的关系,也就是记述了各个基因译码含义。比如,基因 6 的译码含义是 4(二进制译码)。经过逆转操作后,基因 6 虽然被移动,但它的译码含义因仅和其命名或编号有关,所以仍为 4(而不是 16)。如此对扩展表示的个体 A 施加逆转操作后,生成的个体 A' 的基因被重新排序但其适应度值依然保持与原个体一致。

Goldberg 等对此种逆转算子进行了深入的研究[1],并把它与交叉操作有机结合形成了一类独特的交叉操作。这些交叉操作(PMX, CX, OX)在 TSP 问题的求解中效果很好。请读者参阅第三章中有关 TSP 问题求解的介绍。

3. 自适应变异算子(adaptive mutation operator)。

该算子与基本变异算子的操作内容类似,唯一不同的是交叉概

率 P_m 不是固定不变而是随群体中个体的多样性程度而自适应调整。一般是根据交叉所得两个新个体的海明距离进行变化。海明距离越小, P_m 越大, 反之 P_m 越小。

2.10 收敛性

我们已就遗传算法中的编码、适应度函数、选择、交叉和变异等主要操作的基本内容及设计进行了详细的介绍。作为一种搜索算法, 遗传算法通过对这些操作的适当设计和运行, 可以实现兼顾全局搜索和局部搜索的所谓均衡搜索, 具体实现见图 2.22 所示。

应该指出的是, 遗传算法虽然可以实现均衡的搜索, 并且在许多复杂问题的求解中往往能得到满意的结果, 但是该算法的全局优化收敛性的理论分析尚待解决。目前普遍认为, 标准遗传算法并不保证全局最优收敛。但是, 在一定的约束条件下, 遗传算法可以实现这一点。下面, 我们将讨论两个问题:

(1) 未成熟收敛及对策。

(2) 标准遗传算法不具备全局优化收敛性, 但对标准遗传算法的选择算子作一定的修正后, 可以确保收敛性。

2.10.1 未成熟收敛

未成熟收敛是遗传算法中不可忽视的现象, 它主要表现在两个方面:

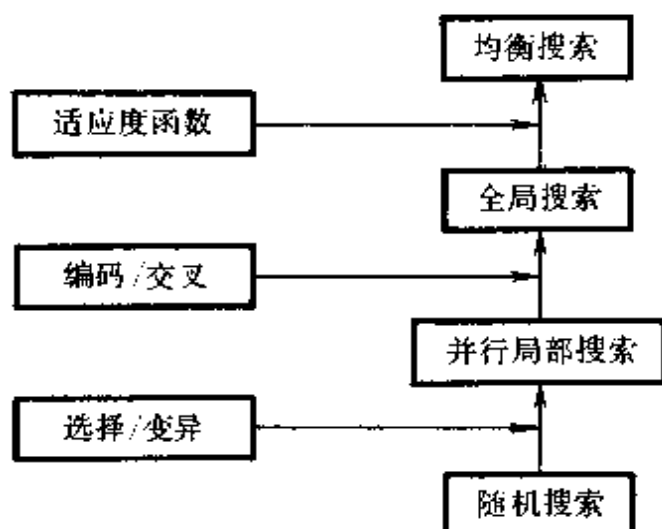


图 2.22 遗传算法均衡搜索的实现

(1) 群体中所有的个体都陷于同一极值而停止进化。

(2) 接近最优解的个体总是被淘汰,进化过程不收敛。

在遗传算法的处理过程的每个环节都有可能导入产生未成熟收敛的因素,具体表现如下:

(1) 在进化初始阶段,生成了具有很高适应度的个体 X ;

(2) 在基于适应度比例的选择下,其它个体被淘汰,大部分个体与 X 一致;

(3) 相同的两个个体实行交叉,从而未能生成新个体;

(4) 通过变异或逆转所生成的个体适应度高但数量少,所以被淘汰的概率很大;

(5) 群体中的大部分个体都处于与 X 一致的状态。

针对上述情况,需要在编码、适应度函数和遗传操作等设计中考虑抑制未成熟收敛的对策。我们曾在第一章和第二章的有关各节中对这些对策作了介绍。这里,我们对这些对策或经验加以总结和扩充:

1) 提高变异概率:在进化初始阶段,它可以加强遗传算法的随机搜索能力。

2) 调整选择概率:可以把选择概率本身也作为个体来进行优化,这就是所谓元 GA (meta GA) 方法。

3) 合适定标:请参阅本章第 2.8.2 节“适应度函数定标”的有关的论述。

4) 维持群体中个体的多样性:为此,a)增加群体规模,但要考虑计算量增加的因素;b)实施局部化:把群体分割成若干子群体,每个子群体独立地进行选择操作,这样可使由于出现不适当个体而产生未成熟收敛的现象局部化;c)实施单一化:把相同的个体单一化,即不允许群体中有若干个相同个体出现;d)增大配对个体距离:配对选择一般是随机进行的,其中缺少对个体间相似度的判断,因此有可能使交叉结果未能产生新个体。作为改进,可用海明距离测度来判断配对个体的相似度,并由此决定配对与否。

上述几种对策的效果各不相同。为了有效地克服未成熟收敛,这些对策也许要在进化过程中分阶段使用或交替使用。

2.10.2 有限马尔柯夫链

在优化理论中,一个算法是收敛的是指算法产生了一个解或函数值的数列,而全局最佳值是该数列的极限值。以下以马尔柯夫链为模型来分析遗传算法的收敛性。首先介绍马尔柯夫链的有关基本概念。

一个有限马尔柯夫链描述了穿越一个有限状态空间 S 的概率轨迹,其中状态数 $|S|=n$ 。在时间步 t ,从状态 $i \in S$ 转换到 $j \in S$ 的概率 $P_{ij}(t)$,称为时间步 t 从 i 到 j 的变换概率。如果变换概率与时间 t 无关,即 $P_{ij}(t)=P_{ij}(S)$,则马尔柯夫链被称为“同构”的。

以 P_{ij} 为元素的矩阵 $P=(P_{ij})$ 称为变换矩阵,其中 $P_{ij} \in [0,1]$,且有:

$$\sum_{j=1}^{|S|} P_{ij} = 1 \quad (i \in S)$$

具有上述性质的矩阵称为随机矩阵。给定一个初始分布 P^0 作为行向量,则该链在 t 个时间步后的分布 $P^t=P^0P^t$ 。因此一个同构的有限马尔柯夫链完全由 (P^0,P) 决定。马尔柯夫链的极限性质取决于变换矩阵的结构。以下给出有关的定义。

定义 2.9 一个矩阵 $A_{n \times n}$ 称为:

① 非负的(nonnegative) ($A \geq 0$),如果对所有 $i,j \in \{1,n\}$,有 $a_{ij} \geq 0$ 。

② 定正的(positive) ($A > 0$),如果对所有 $i,j \in \{1,n\}$,有 $a_{ij} > 0$ 。

定义 2.10 一个非负矩阵 $A_{n \times n}$ 称为:

③ 基本的(primitive),如果存在 $ak \in N$,使得 A^k 为定正。

④ 可归约的(reducible),如果 A 能通过对行和列施加相同的变换,变换成以下形式:

$$A' = \begin{bmatrix} C & 0 \\ R & T \end{bmatrix} \quad (C, T \text{ 为方阵})$$

⑤ 不可归约的(irreducible),如果矩阵不是可归约的。

⑥ 随机的(stochastic),如果对于所有的 $i \in \{1, \dots, n\}$, 有 $\sum_{j=1}^n a_{ij} = 1$ 。

⑦ 稳定的(stable),如果矩阵存在相同行。

⑧ 列可容的(column-allowable),如果在每一列至少存在一个正元素。

可以看出,随机矩阵的乘积还是随机的,所有的定正矩阵都是基本矩阵。以下给出几个重要定理。

引理 2.1 设 C, M, S 是随机矩阵,其中 M 是定正的, S 是列可容的,则 $A = CMS$ 定正的。

[证明] 设 $A = CM, B = AS$, 因为 C 是随机矩阵,则 C 的每一行至少存在一个正元素。因此,对任意 $i, j \in (1, \dots, n)$, 有 $a_{ij} =$

$\sum_{k=1}^n c_{ik} \cdot m_{kj} > 0$, 即 $A > 0$ 。同理,因为 S 是列可容的,则对任意 $i, j \in$

$(1, \dots, n)$, 有 $b_{ij} = \sum_{k=1}^n a_{ik} \cdot s_{kj} > 0$, 即 $B > 0$ 。证毕。

定理 2.2^[13] 设 P 是一个基本随机矩阵,则当 $k \rightarrow \infty$ 时, P^k 收敛于一个定正稳定的随机矩阵 $P^\infty = 1' P^\infty$, 其中 $P^\infty = P^0 \cdot \lim_{k \rightarrow \infty} P^k = P^0 \cdot P^\infty$ 具有非 0 元素并且唯一,同时,与初始分布无关。

定理 2.3^[13] 设 P 是可归约随机矩阵,其中 $C_{m \times m}$ 是一个基本随机矩阵, R 和 T 不为 0, 则

$$P^\infty = \lim_{k \rightarrow \infty} P^k = \lim_{k \rightarrow \infty} \begin{bmatrix} C^k & 0 \\ \sum_{i=0}^{k-1} T^i R C^{k-1-i} & T^k \end{bmatrix} = \begin{bmatrix} C^\infty & 0 \\ R^\infty & 0 \end{bmatrix}$$

是一个稳定的随机矩阵,其中 $P^\infty = 1' P^\infty, P^\infty = P^0 \cdot P^\infty$ 是唯一的,与初始分布无关,且满足 $P_i^\infty > 0, 1 \leq i \leq m; P_i^\infty = 0, m < i \leq n$ 。

以上两定理没有给出证明,有兴趣的读者可参考有关文献。

2.10.3 标准遗传算法的收敛性

标准遗传算法可被描述为一个马尔可夫链:状态空间 $S = \text{IB}^n = \text{IB}^l \cdot n$, 其中 n 表示群体大小, l 表示串个体的长度。状态空间的每个元素可以被认为是一个二进制表示的整数。映射 $\pi_k(i)$ 提取状态 i 的二进制表示中的第 k 个长度为 l 的段, 用于区分个体和群体。

群体中各基因(个体中的一位)由于遗传算子引起的概率变化由变换矩阵 P 描述, 而 P 又可以很自然地分解为随机矩阵 C, M, S 的乘积, 即有 $P = CMS$ 。其中 C 描述交叉算子引起的变化, M 描述变异算子的作用, S 则描述选择算子引起的变化。

定理 2.4 如果变异概率为 $P_m \in (0, 1)$, 交叉概率为 $P_c \in [0, 1]$, 同时采用比例选择法(按个体适应度占群体适应度的比例进行复制), 则标准遗传算法的变换矩阵 P 是基本的。

[证明] 交叉算子可以被认为是在状态空间 S 中的一个随机满射函数, 即 S 的每个状态都随机地映射到另一状态。因此 C 是随机矩阵。同理, M, S 也都是随机矩阵。因为变异算子独立地施加到群体中的每个基因(位), 因此在变异算子作用下, 从状态 i 变成状态 j 的概率总计为 $M_{ij} = P_{m,ij}^{H_{ij}} (1 - P_m)^{N-H_{ij}} > 0$, 其中 H_{ij} 表示状态 i 和状态 j 的海明距离, 因此 M 是定正的。

选择算子不改变变异算子产生的状态的概率的下界为:

$$S_{ii} \geq \frac{\sum_{k=1}^n f(\pi_k(i))}{\left(\sum_{K=1}^n F(\pi_K(i)) \right)^n} > 0$$

因此 S 是列可容的。

由引理 2.1, 则 $P = CMS$ 是定正的, 而每个定正矩阵都是基本的。证毕。

推论 2.1 具有如定理 2.4 一样的参数的标准遗传算法是一个遍历马尔可夫链, 即对于任意初始分布, 在任意时刻、任意状态的概率非零的链存在唯一的极限分布。

我们注意到初始分布 P^0 对于马尔可夫链的极限性质没有任何影响。因此从理论上看来,算法可以任意初始化,进入循环前的选择也可以忽略。

遍历性对于标准遗传算法的收敛性有很大的影响,为避免混淆,我们给出遗传算法收敛的精确定义。

定义 2.11 设 $Z_t = \max\{f(\pi_k^{(t)}(i)); k=1, \dots, n\}$ 是一个随机变量序列,该变量代表在时间步 t 状态中最佳的适应度。遗传算法收敛到全局最佳值,当且仅当:

$$\lim_{t \rightarrow \infty} P\{Z_t = f^*\} = 1$$

其中 $f^* = \max\{f(b) | b \in IB'\}$, 即全局最佳值。

定理 2.5 标准遗传算法(参数如定理 2.4)不能收敛至全局最佳值。

[证明] 设 i 是状态空间的一个状态,其中 $\max\{f(\pi_k(i)) | k=1, \dots, n\} < f^*$, P_t^i 是时间步 t 遗传算法进入状态 i 的概率。显然, $P\{Z_t \neq f^*\} \geq P_t^i \Leftrightarrow P\{Z_t = f^*\} \leq 1 - P_t^i$ 。由定理 2.2,遗传算法在状态 i 的概率收敛至 $P_i^\infty > 0$ 。则有

$$\lim_{t \rightarrow \infty} P\{Z_t = f^*\} \leq 1 - P_i^\infty < 1$$

因此,标准遗传算法是不收敛的。证毕。

由定理 2.5,可以知道,具有变异概率 $P_m \in (0, 1)$, 交叉概率 $P_c \in [0, 1]$ 以及按比例选择的标准遗传算法是不能收敛至全局最佳值。这无疑是一个令人沮丧的结论。然而,庆幸的是,只要对标准遗传算法作一些改进,就能够保证其收敛性。下面讨论这个问题。

我们对标准遗传算法作一定改进,即不按比例进行选择,而是保留当前所得的最佳值(称作超个体)。对马尔可夫链的描述也作一定的扩展,即该超个体不参与遗传作用。状态空间 S 的基从 2^{n-1} 扩展至 $2^{(n-1)+1}$ 。为了描述方便,将该超个体置于最左的位置,并能用 $\pi_0(i)$ 从状态 i 的群体加以访问。那些包含相同超个体的状态的变换概率假设在变换矩阵呈阶梯排列,并且超个体的适应度越高,其相应状态的

位置也越高。

由于最佳个体不受遗传算子作用,扩展的 C^+, M^+, S^+ 可被写成块对角矩阵,即

$$C^+ = \begin{bmatrix} C & & & \\ & C & & \\ & & \ddots & \\ & & & C \end{bmatrix}, M^+ = \begin{bmatrix} M & & & \\ & M & & \\ & & \ddots & \\ & & & M \end{bmatrix}$$

$$S^+ = \begin{bmatrix} S & & & \\ & S & & \\ & & \ddots & \\ & & & S \end{bmatrix}$$

其中 C, M, S 为 2^d 方阵, C^+, M^+, S^+ 为 $2^{(n+1)d}$ 方阵, 则有

$$C^+ M^+ S^+ = \begin{bmatrix} CMS & & & \\ & CMS & & \\ & & \ddots & \\ & & & CMS \end{bmatrix}$$

其中 $CMS > 0$ 。

选择操作由“改良”(upgrade)矩阵 U 表示,其作用是将一个包含优于当前超个体的个体的中间状态变换至超个体等于该个体的状态。特别地,设 $b = \arg \max \{f(\pi_k(i)) | k=1, \dots, n\} \in IB^i$ 表示除超个体外,任意状态之中的最佳个体。如果 $f(\pi_0(i)) < f(b)$, 则 $U_{ij} = 1$, 其中 $j^{\text{def}}(b, \pi_1(i), \dots, \pi_n(i), \pi_n(i)) \in S$; 否则 $U_{ij} = 0$ 。这样,每一行中恰好只有一个元素。但对于列该结论并不成立,因为对于 S 中每个状态

$j, f(\pi_0(j)) < \max\{f(\pi_k(j)) \mid k=1, \dots, n\}$ 使得 $U_{ij}=0$ 。换句话说, 一个状态要么被改进, 要么保持不变。因此, 改良矩阵的结构可写成:

$$U = \begin{bmatrix} U_{11} & & & & \\ U_{21} & U_{22} & & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & & \cdot & \\ \cdot & \cdot & & & \cdot \\ U_{2^l,1} & U_{2^l,2} & \cdot & \cdot & \cdot & U_{2^l,2^l} \end{bmatrix}$$

其中 $U_{a,b}$ 大小为 $2^{n^l} \times 2^{n^l}$ 。为了简单起见, 假设只有一个全局最佳解。则只有 U_{11} 为单位矩阵, 而其它所有 $U_{aa} (a \geq 2)$ 是单位矩阵且具有对角零值。 $P = CMS$, 则有:

$$\begin{aligned} P^+ &= \begin{bmatrix} P & & & & \\ & P & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & P \end{bmatrix} \begin{bmatrix} U_{11} & & & & \\ U_{21} & U_{22} & & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & & \cdot & \\ \cdot & \cdot & & & \cdot \\ U_{2^l,1} & U_{2^l,2} & \cdot & \cdot & \cdot & U_{2^l,2^l} \end{bmatrix} \\ &= \begin{bmatrix} PU_{11} & & & & \\ PU_{21} & PU_{22} & & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & & \cdot & \\ \cdot & \cdot & & & \cdot \\ PU_{2^l,1} & PU_{2^l,2} & \cdot & \cdot & \cdot & PU_{2^l,2^l} \end{bmatrix} \end{aligned}$$

其中 $PU_{11} = P > 0$ 。子矩阵 $PU_{a1} (a \geq 2)$ 可以集中在一起以长方矩阵 $R \neq 0$ 表示, 则有如下定理:

定理 2.6 具有定理 2.4 所示参数, 且在选择后保留当前最优值的遗传算法最终能收敛到全局最佳值。

[证明] 子矩阵 $PU_{11} = P > 0$ 中集中了包含全局最佳个体的状态(称为全局最优状态)的变换概率。因为 P 是一个基本随机矩阵且

$R \neq 0$, 则定理 2.3 保证了所有包含在非全局最优状态中的概率收敛于 0。则所有包含在全局最优状态中的概率收敛于 1, 因此有

$$\lim_{t \rightarrow \infty} P\{Z_t = f^*\} = 1$$

即算法收敛于全局最优解。证毕。

当然, 在选择算子作用后保留当前最优解是一项比较复杂的工作, 因为该解在选择算子作用后可能丢失。但是定理 2.6 至少表明了这种改进的遗传算法能够收敛至全局最优解。有意思的是, 实际上只要在选择前保留当前最优解, 就可以保证收敛, 定理 2.7 描述了这种情况。

定理 2.7 具有定理 2.4 参数的, 且在选择前保留当前最优解的遗传算法可收敛于全局最优解。

[证明] 变换矩阵 $P^+ = T^+ U S^+$, 其中 $T^+ = C^+ M^+$, 设 $T = CM$, 则有

$$P^+ = \begin{bmatrix} T & & & & \\ & T & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & T \end{bmatrix} \begin{bmatrix} U_{11} \\ U_{21} & U_{22} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ U_{2',1} & U_{2',2} & \cdot & \cdot & \cdot & U_{2',2'} \end{bmatrix}$$

$$\times \begin{bmatrix} S & & & & \\ & S & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & S \end{bmatrix}$$

$$= \begin{bmatrix} TU_{11}S \\ TU_{21}S & TU_{22}S \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ TU_{z',1}S & TU_{z',2}S & \cdot & \cdot & \cdot & TU_{z',z'}S \end{bmatrix}$$

其中 $TU_{11}S = TS = P > 0$, 同样子矩阵 $TU_{a1}S (a \geq 2)$ 集中在一起以长方形矩阵 $R \neq 0$ 表示。同定理 2.6, 则算法可收敛于全局最优解。证毕。

综上所述, 我们看到标准的遗传算法(交叉概率为 p_c , 变异概率为 p_m , 比例选择法)在任意初始化, 任意交叉算子(简单的或复杂的)以及任意适应度函数下, 都不能收敛至全局最优解。而通过改进遗传算法, 即在选择作用前(或后)保留当前最优解, 则能保证收敛至全局最优解。由此可知, 收敛至全局最优解, 实际上是由于不断地保留当前最优解的过程的结果。

需要指出的是, 尽管证明了改进的遗传算法最终能收敛至全局最优解, 但收敛到最优解所需的时间可能是很长的。关于遗传算法最终收敛至全局最优解的时间复杂度仍是有待解决的问题。

参 考 文 献

- [1] Holland J H. Adaptation in Natural and Artificial Systems. The University of Michigan Press, 1975
- [2] Goldberg D E. Genetic Algorithms in Search, Optimization & Machine learning. Addison-Wesley Publishing, 1989
- [3] Holland J H. Genetic Algorithms and the Optimal Allocations of Trials. SIAM Journal of Computing, 1973, 2(2): 88 ~ 105
- [4] Bagley J D. The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms. Dissertation Abstracts International, 1967, 28(12), 5106B (University Microfilms No. 68-7556)
- [5] Rosenberg R S. Simulation of Genetic Populations with Biochemical Properties. Dissertation Abstracts International, 1967, 28(7), 2732B (University Microfilms 67-17,836)
- [6] Bethke A D. Genetic Algorithms as Function Optimizers. Dissertation Abstracts International, 1981, 41(9), 3503B (University Microfilms No. 8106101)
- [7] Holland J H. Genetic Algorithms and Classifier Systems: Foundations and Future Directions. Genetic Algorithms and Their Applications : Proceedings of the Second International Conference on Genetic Algorithms. 1987. 82~89
- [8] 王丽薇. 遗传算法的研究与应用:[博士学位论文]. 哈尔滨工业大学, 1994
- [9] Goldberg D E. Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning, Part I: Rule Learning Control of a Pipeline under Normal and Abnormal

Conditions. Engineering with Computers, 1987; 47~58

- [10] Bridges G L, Goldberg. D E An Analysis of Reproduction and Crossover in a Binary-coded Genetic Algorithm. Genetic Algorithms and Their Applications; Proceedings of the Second International Conference on Genetic Algorithms. 1987. 9~13
- [11] Goldberg D E. Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning, Part 1: Genetic Algorithms in Pipeline Optimization. Engineering with Computers. 1987. 35~45
- [12] Goldberg D E. Optimal Initial Population Size for Binary-coded Genetic Algorithms (TCGA Report No. 85001). University of Alabama, The Clearing-house for Genetic Algorithms, 1985
- [13] Iosifescu M. Finite Markov Processes and Their Applications. Willey, 1980
- [14] 山村, 小野, 小林, 形質の遺伝を重視した遺伝的アルゴリズムに基づく巡回セールスマン問題の解法。人工知能学会誌, 7-6, 965~969, 1992
- [15] Jomikow C Z, Michalewicz Z. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithm. Proc of ICGA, 1991. 31~36
- [16] Goldberg D E, Deb K, korb B. Don't Worry Be Messy. Proc of ICGA, 1991. 24~30
- [17] Louis S J, Rawlins G J E. Design Genetic Algorithm; Genetic Algorithm in Structure Design. Proc of ICGA, 1991
- [18] Cartwright H M and Mott G F. Looking Around; using Clues from the Data Space to Guide Genetic Algorithm Searches. Proc of ICGA, 1991. 108~114

- [19] Das R, whitley D. The Only Challenging Problems are Deceptive; Global Search by Solving Order-1 Hyperplanes. Proc of ICGA, 1991. 166~173
- [20] Holland J. Adaptation in Nature and Artificial System. MIT Press, 1991
- [21] Goldberg D E. Genetic Algorithm and Walsh Functions, Part I & II. Complex Systems, 1989(3)
- [22] 山村, 織田, 小林. マルコフ過程によるSimple GAの解析、日本機械学会第2回FAMシニボジウム講演論文集。1992. 283~388

第三章 遗传算法与组合优化

组合优化(combinatorial optimization)是遗传算法最基本的也是最重要的研究和应用领域之一。所谓组合优化是指在离散的、有限的数学结构上,寻找一个满足给定约束条件并使其目标函数值达到最大或最小的解。一般来说,组合优化问题通常带有大量的局部极值点,往往是不可微的、不连续的、多维的、有约束条件的、高度非线性的 NP 完全问题,因此,精确地求解组合优化问题的全局最优解一般是不可能的。遗传算法作为一种新型的、模拟生物进化过程的随机化搜索、优化方法,近十几年来在组合优化领域得到了相当广泛的研究和应用,并已在解决诸多典型组合优化问题中显示了良好的性能和效果。本章首先简要介绍基于遗传算法进行组合优化的基本策略和一些进一步的实用技术,然后具体叙述遗传算法在解决背包问题、图的划分问题以及货郎担问题等方面的几个典型应用。

3.1 基于遗传算法的组合优化方法

采用遗传算法的组合优化方法可用算法 3.1 一般性地描述。

算法 3.1 遗传优化的一般方法

(1) 确定群体规模 n (整数),使用随机方法或其它方法产生 n 个可能解 $X_i(k)$ ($1 \leq i \leq n$) 组成初始解群。

(2) 对于每一个个体 $X_i(k)$ (变量 k 称作“代”数,初始时 $k=1$), 计算其适应度 $f(X_i(k))$ 。

(3) 对于每一个体 $X_i(k)$, 计算其生存概率 $P_i(k)$:

$$P_i(k) = f(X_i) / \sum_{i=1}^n f(X_i)$$

然后,设计一个随机选择器,依据 $P_i(k)$ 以一定的随机方法产生配种个体 $X_i(k)$ 。

(4) 产生下一代解群。选取两个配种个体 $X_1(k)$ 、 $X_2(k)$, 并依据一定的组合规则(如交叉、变异、逆转等)将 $X_1(k)$ 、 $X_2(k)$ 结合成两个新一代的个体 $X_1(k+1)$ 、 $X_2(k+1)$, 直至新一代 n 个个体形成完毕。

(5) 重复(2)–(4)步,直至满足程序终结条件(如时间上的限制或解的质量达到满意的范围等)。

上述算法在合适的条件下,其目标函数值会逐代增加,并收敛到某一最大值。算法 3.1 涉及的主要问题有编码方案、适应度函数设计、约束条件处理、选择机制、遗传操作,遗传算法参数确定等。在组合优化问题的实际应用中,合理地处理以上问题,构造合适的遗传算法框架是遗传优化的关键所在。由于第二章已经就最基本的遗传算法优化技术作了比较详细的描述,为了完整起见,这里仅对遗传算法的关键参数确定简单提及一下,然后对有关遗传操作的更进一步的流行技术展开一些讨论。

3.1.1 遗传算法的关键参数确定

遗传算法的关键参数主要包括以下参数。

1. 群体规模 n

群体规模影响遗传优化的最终结果以及遗传算法的执行效率。当群体规模 n 太小时,遗传算法的优化性能一般不会太好,而采用较大的群体规模则可减少遗传算法陷入局部最优解的机会,但较大的群体规模意味着计算复杂度高。一般取 n 从 10 到 160 之间。

2. 交叉概率 P_c

交叉概率 P_c 控制着交叉操作被使用的频度。较大的交叉概率可增强遗传算法开辟新的搜索区域的能力,但高性能的模式遭到破坏的可能性增大;若交叉概率太低,遗传算法搜索可能陷入迟钝状态。一般取 P_c 从 0.25 到 1.00 之间。

3. 变异概率 P_m

变异在遗传算法中属于辅助性的搜索操作,它的主要目的是维持解群体的多样性。一般,低频度的变异可防止群体中重要的、单一基因的可能丢失,高频度的变异将使遗传算法趋于纯粹的随机搜索。通常取变异概率 P_m 为 0.001 左右。

4. 代沟 G (generation gap)

代沟 G 控制着每一代群体构成中个体被更新的百分比,即在 t 代有 $n(1 - G)$ 个个体结构选择复制到 $t+1$ 代解群体中。“代沟”方式的选用为遗传算法利用优化过程的历史信息提供了条件,加速了遗传算法的收敛过程;当代沟 G 过小时,可能导致遗传算法的过早的不成熟的收敛。一般取 G 从 0.30 到 1.00。

3.1.2 几种流行的选择机制

在组合优化过程中,选取合适的选择机制也是有效解决组合优化问题的关键技术之一。目前流行的遗传算法选择机制主要有以下几种:

1. 赌轮选择(roulette wheel selection)机制

这是一种最基本的、常用的选择机制。在这种选择机制中,个体每次被选中的概率与其在群体环境中的相对适应度成正比(详见第二章)。

2. 最佳保留(elitist model)选择机制

首先按赌轮选择机制执行遗传算法的选择功能;然后将当前解群体中适应度最高的个体结构完整地复制到下一代群体中。其主要优点在于能保证遗传算法终止时得到的最后结果一定是历代出现过的最高适应度的个体(详见第二章)。

3. 期望值模型(expected value model)选择机制

这种选择机制主要是为了克服赌轮选择机制所产生的随机选择错误而提出的。在赌轮选择机制中,由于群体规模有限等原因,使得个体实际被选中的次数与它应被选中的期望值($n \cdot f(X_i) / \sum_{i=1}^n f(X_i)$)之间可能存在着一定的误差。在期望值模型选择机制中,

先按期望值($n \cdot f(X_i) / \sum_{i=1}^n f(X_i)$)的整数部分安排个体被选中的次数,而对其期望值($n \cdot f(X_i) / \sum_{i=1}^n f(X_i)$)的小数部分可按下述几种方式进行处理:

(1) 确定方式

将期望值($n \cdot f(X_i) / \sum_{i=1}^n f(X_i)$)小数部分按值的大小进行列表,然后自大到小依次选择,直到选满为止。

(2) 赌轮方式

将期望值($n \cdot f(X_i) / \sum_{i=1}^n f(X_i)$)的小数部分按上述的赌轮选择机制进行选择,直到选满为止。

(3) 贝努利试验(Bernoulli Trials)方式

将期望值($n \cdot f(X_i) / \sum_{i=1}^n f(X_i)$)的小数部分作为概率进行贝努利试验,若试验成功,则该个体被选择,如此反复试验,直到选满为止。例如,一个个体被选择的期望值为 1.49,那么这个个体肯定有一次被选择的机会,是否还有第二次被选择的机会就要看依概率 0.49 进行的贝努利试验是否成功,若成功则该个体还要被选择一次,若贝努利试验失败则该个体就没有第二次被选择的机会。

4. 随机竞争(stochastic tournament)选择机制

这种机制与赌轮选择机制差不多。不过,在随机竞争选择机制中,每次按赌轮选择机制选取一对个体,然后让这两个个体进行竞争,适应度高者获胜,如此反复,直到选满为止。

5. 排序(ranking)选择机制

上述几种选择机制虽各有特点,但核心内容基本相同,均直接以适应度为基础进行比例选择,个体被选择的次数与其适应度大小成正比。这种直接基于个体适应度函数值的选择机制在遗传算法优化过程中可能潜伏着以下两个问题:

(1) 当群体中出现个别或极少数适应度相当高的染色体时,这

类选择机制可能导致个别或极少数染色体基因在群体中迅速地遗传,引起解群体的不成熟收敛。

(2) 另一方面,当群体内的个体适应度彼此非常接近的时候,这类选择机制将趋向纯粹的随机选择,从而使优化过程陷于迟钝状态。这种情形在遗传算法优化的后期经常出现。

解决上述问题的一种途径是采用第二章所述的适应度函数的动态定标方案,另一种常用的方法便是下面将要详述的排序选择机制。

排序选择机制是一种基于群体内各个体之间的相对适应度设计的等级选择机制。它首先依据各个体的适应度大小进行排序。适应度最大的安排在第一号,最小的安排在第 n 号(设群体规模等于 n);然后基于所排序号按某种规则进行选择,排在前面的个体有较多的被选择的机会。算法 3.2 描述了一种线性函数选择方案,其中,偏置 bias 对选择强度有影响,当 $\text{bias}=2.0$ 时,第一号个体被选择的次数大致等于正中间个体被选择次数的两倍。

算法 3.2 排序选择算法

`linear()`

```
float bias=2.0;int index;double sqrt;  
index=POPULATION_SIZE*(bias-  
    sqrt(bias*bias-4.0(bias-1)*random()))/  
    2.0/(bias-1);  
return(index);
```

`random()`返回自0.00到1.00之间的随机数。

排序选择机制可以较好的处理比例选择机制存在的上述两个问题,其缺点是选择标准过于偏离个体的适应度。

6. 混合选择机制

将上述几种基本选择机制进行有限的混合在解决复杂的组合优化问题时,有可能在整体上改善算法的优化性能。

3.1.3 适应度函数的定标

所谓适应度函数的定标(fitness scaling)是指对目标函数值的某种映射变换,有关此内容已在第二章进行了详细讨论,其主要方法再归纳如下:

1. 线性定标(scaling)

$$f(z) = az + b$$

其中,常量 a 和 b 是预先确定的。

2. 幂函数定标

$$f(z) = z^{\alpha}$$

其中 α 为一常量,其取值需要在理性分析估计的基础上,结合一些实验进行一定程度的精细调整才能获得较好的结果。

3. 指数定标

$$f(z) = \exp(-bz)$$

这种机制是受模拟退火方法的启发得出的。

选取合适的适应度定标方案是非常重要的。首先,遗传算法的选择强度可以通过适应度函数值的伸、缩加以控制。一般来说,在遗传优化的初始阶段希望选择强度稍低一些,以免遗传群体被单个或少几个适应度较高的个体所支配;在遗传优化的后期,即算法接近收敛的情况下,由于群体内的个体间适应度的差异较小,继续优化的潜能较低,因此有必要适度地提高选择强度,以便遗传算法收敛到一更优的解。其次,实际应用中常碰到目标函数值小于0的情况,此时若直接使用目标函数值作为适应度将会引起“适应度”概念理解上的问题及随机选择方面的问题,因此使用适应度函数定标技术将目标函数值域映射到正实数值域。此外,使用适应度定标技术还可处理那些无目标函数值的组合优化问题,在此情况下,我们只能获取个体性能之间相对好坏的概念,如个体1比个体2好(或坏),而不能计算或特别难以计算个体的目标函数值,因而我们只能依据群体内个体性能的相对优劣,以一定的函数映射方法对个体性能进行适应度定标。当

然,解决无目标函数值的组合优化问题还可选用前面提到的排序(rank)选择机制,这实质上也是一种适应度定标方案。

3.1.4 二倍体(diploidy)与显性(dominance)技术

1. 二倍体与显性

所谓二倍体是指基因型带有两套同型染色体。在自然界中,单倍体(haploid)这种最简单的染色体结构形式多出现在不太复杂的低级生物体中,而高级生物体中一般含有更为复杂的染色体结构,如二倍体或多倍体(polyploid)结构。为何高级生物采取这种多套同型染色体结构编码相同的功能信息?更进一步讲,一旦这些多倍体结构译码为不同的功能,生物选择哪种基因作为特征表现?对于二倍体这种相对比较简单的染色体结构,显性是一种被实验证实的译码方式。例如下面的二倍体结构染色体:

AbCDe

aBCde

每个字母代表一个等位基因,大写字母表示显性等位基因,小写字母表示隐性(recessives)等位基因。根据生物学的显性规则,上述二倍体的表现型(phenotype)为:

ABCDe

在每个位点,我们看到显性基因总是被表现,而隐性基因仅当两个同型染色体中等位基因皆为隐性基因时才能被表现。用遗传学的术语来说,显性基因在纯合子(AA)或杂合子(Aa 或 aA)情况下均能被表现,而隐性基因只能在纯合子(aa)的情况下才能被表现。更抽象一点,我们可用从基因型到表现型的一种函数映射来描述显性现象。

虽然二倍体及显性的机理相对来说比较清晰,但生物为何采用这种明显冗余的信息编码结构及显性算子的屏蔽保护技术却是遗传学的一个难题。一种对人工遗传搜索比较有价值的遗传学解释是:二倍体提供了记忆曾经有用的基因及基因组合的一种机制,显性提供了一种保护被记忆的基因免受有害选择所破坏的算子。在自然界,生

态环境常发生或大或小、或好或坏、或快或慢的变化,因此,有生命力的生物应能迅速地适应环境的变化。二倍体结构所提供的冗余记忆能力使得生物不易忘记在以前的环境变化过程中所学会的知识,而显性或显性的变化能唤醒对旧知识的记忆,偶尔试验一下旧知识的效果,因而能表现更强的自适应环境的能力。生物学家已经证实显性也能进化,基因的显性与非显性受遗传控制。下面将要介绍显性与二倍体在人工遗传搜索中的应用。

2. 遗传算法中的二倍体与显性技术

自60年代以来,已有不少学者研究二倍体与显性技术 (Bagley^[1], Rosenberg^[2], Hollstien^[3], Brindle^[4], Glodberg & Smith^[5]等)。1971年,Hollstien 研究了两种二倍体与显性技术的实现方案,其一是二进制的两位点显性机制,另一种是三进制的单位点显性机制。在两位点显性机制中,基因由功能基因和修饰基因两部分组成;功能基因由数值0 或1 表示,其值反映参数的编码状态;修饰基因用 M 和 m 表示。在这种机制中,当二倍体的对应的两个修饰基因中至少有一个为 M 时,0 等位基因才变为显性;整个显性映射关系如表3.1 所示。

从映射关系的本质来看,这种二进制的两位点显性机制也可用一种三进制的单位点显性机制所替代。在单位点显性机制中,Hollstien 选用了0,1,2 三个数字表示等位基因;其中,2 扮演了显性“1”的角色,1 扮演了隐性“1”的角色,其基因型到表现型的映射关系如表3.2 所示。

表 3.1 两位点显性映射

| | 0M | 0m | 1M | 1m |
|----|----|----|----|----|
| 0M | 0 | 0 | 0 | 0 |
| 0m | 0 | 0 | 0 | 1 |
| 1M | 0 | 0 | 1 | 1 |
| 1m | 0 | 1 | 1 | 1 |

表 3.2 单位点三进制显性映射

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 1 |

Hollstien 曾在函数优化研究中使用了这种二进制单位点显性技术^[3]。实验结果表明,与常规的单倍体遗传算法相比,二倍体编码方式能较好地维持遗传群体的多样性,但在群体的平均适应度及优化的最终结果方面没有明显的改进。后来的一些研究显示,二倍体及显性技术可能仅适用于环境变化的场合。1987年,Smith 和 Goldberg^[6]比较了单倍体与二倍体技术在“非稳定背包问题”优化方面的性能差异。所谓“非稳定背包问题”是指参数随时间变化的背包问题。在 Smith 和 Goldberg 的实验研究中,背包容量 C 以周期 T 在 C_0 和 C_1 之间不断变化,分别使用单倍体和二倍体技术对这一问题进行优化,实验结果如图 3.1~3.4 所示。从图中可以看出,单倍体遗传算法几乎完全不能跟踪问题环境的变化,而基于二进制单位点显性技术的二倍体遗传算法则显示了较好的适应问题环境变化的能力。

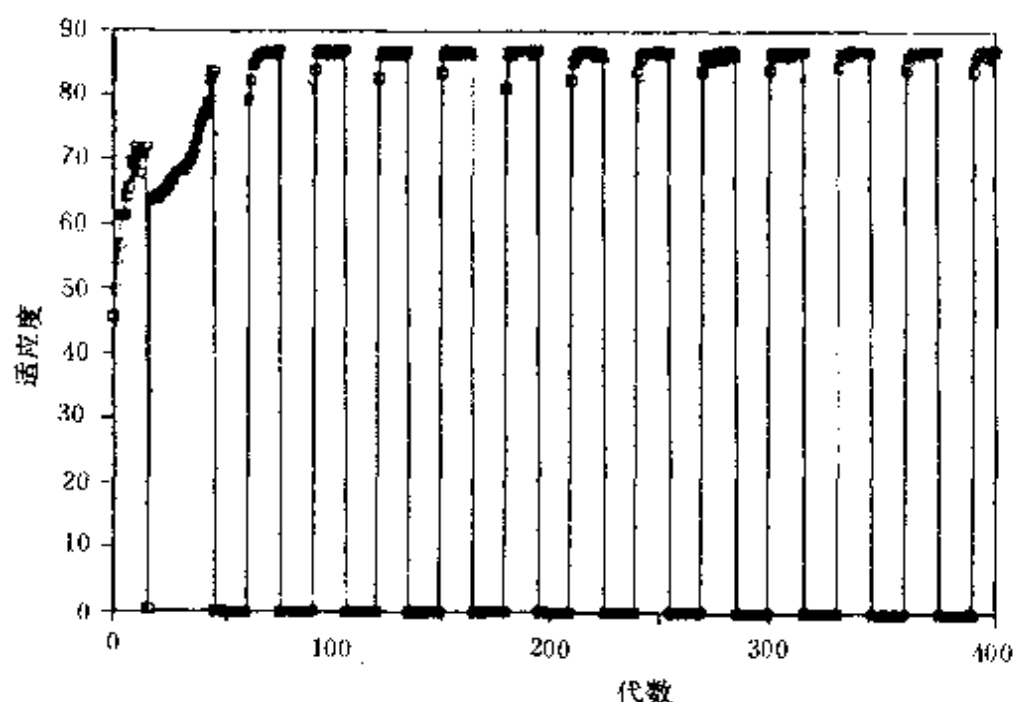


图 3.1 基于单倍体遗传算法求解非稳定背包问题的平均适应度曲线

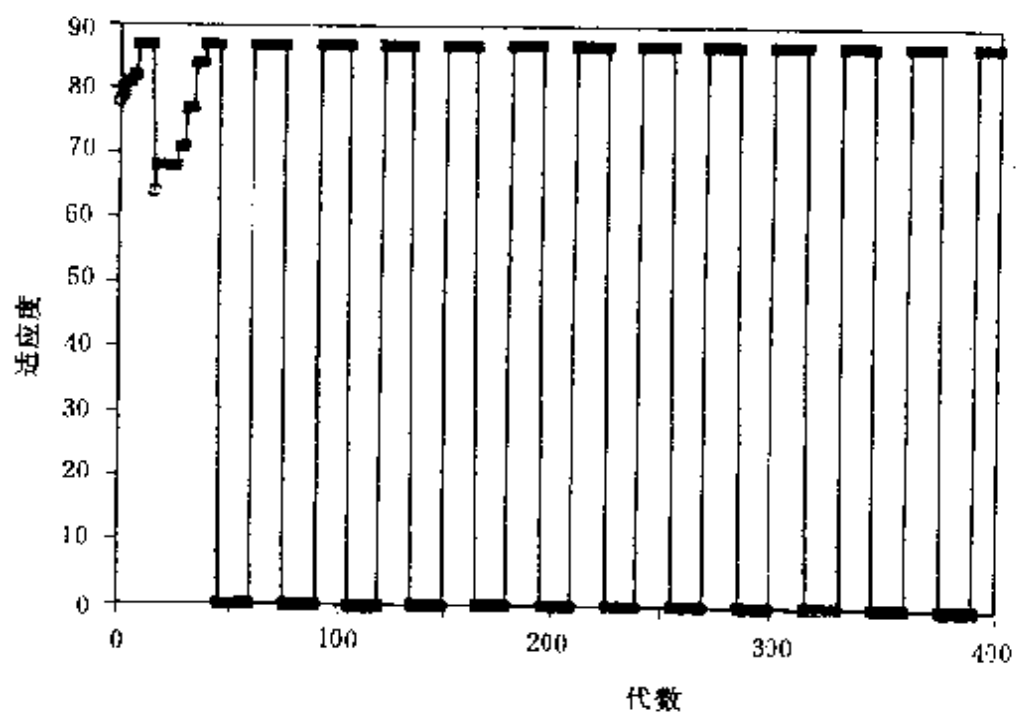


图 3.2 基于单倍体遗传算法求解非稳定背包问题的最佳适应度曲线

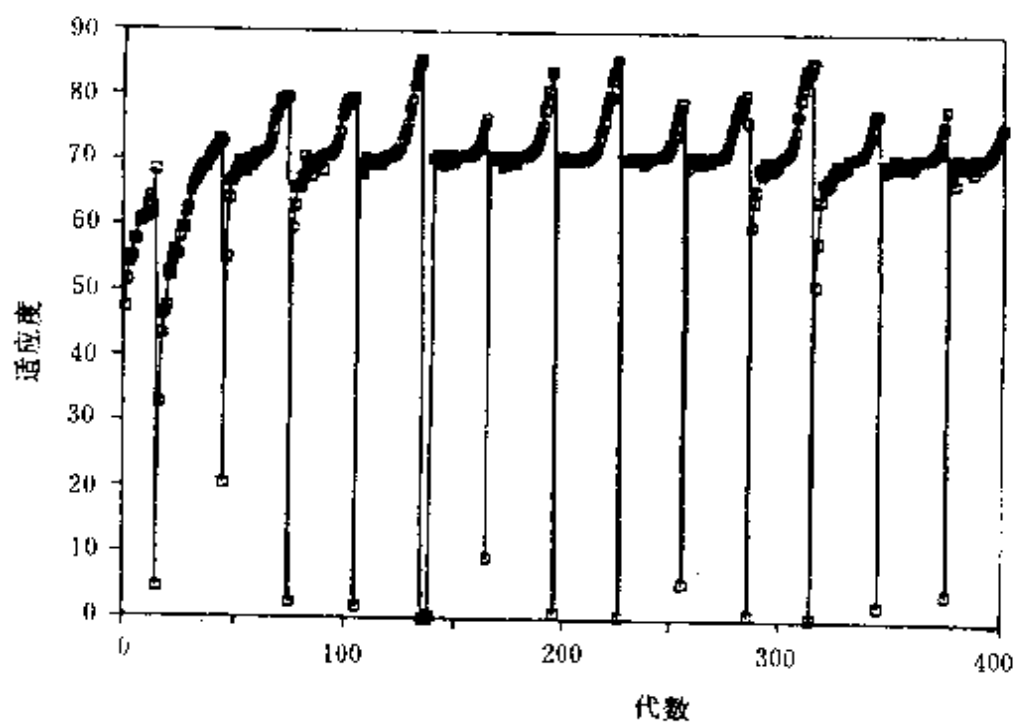


图 3.3 基于二倍体遗传算法求解非稳定背包问题的平均适应度曲线

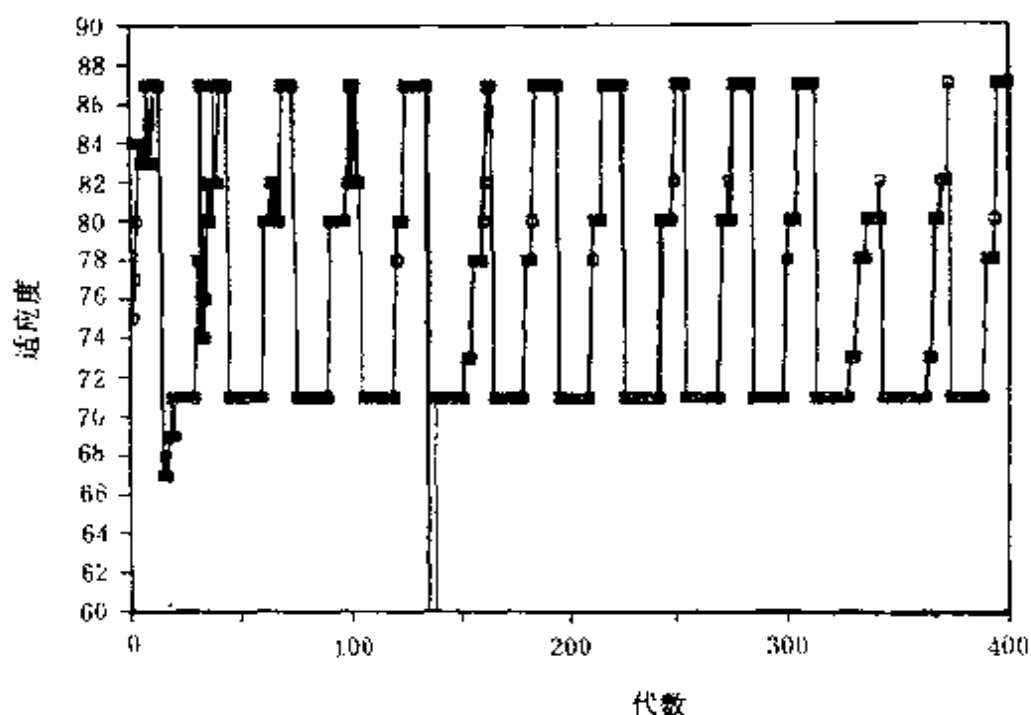


图 3.4 基于二倍体遗传算法求解非稳定背包问题的最佳适应度曲线

3.1.5 物种形成(speciation)与小生境(niche)技术

在自然界,“人以群分,物以类聚”是一个司空见惯的现象。生物总是倾向于与自己特征、性状相类似的生物(同类)生活在一起,一般总是与同类交配繁衍后代。这种正选型交配方式在生物遗传进化过程中是有其积极的作用的。生物学上,小生境(niche)是指特定环境中一种组织(organism)的功能,而把有共同特性的组织称作物种(species)。

为了理解分类及小生境技术在遗传优化中的作用,我们先考察一下基本遗传算法在单变量多峰函数优化方面的搜索特性。图3.5表示的是一个等峰的单变量多极值点函数。如果我们以均匀分布的随机方式产生初始群体,则在算法的开始阶段,各个体分布在一个相对较宽的函数定义域中;随着遗传优化过程的进展,群体开始爬山,并逐步集中到一个山峰上。这种收敛到一个山峰(或山谷)的优化特点是由群体规模有限所引起的随机采样错误造成的。对于图3.6所示的

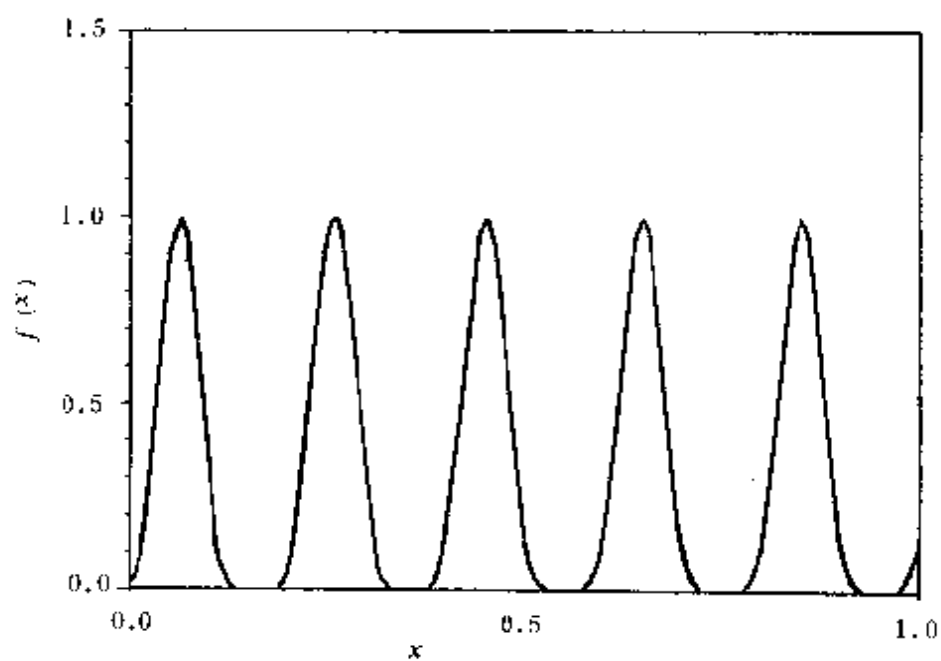


图 3.5 等峰的单变量多极值点函数

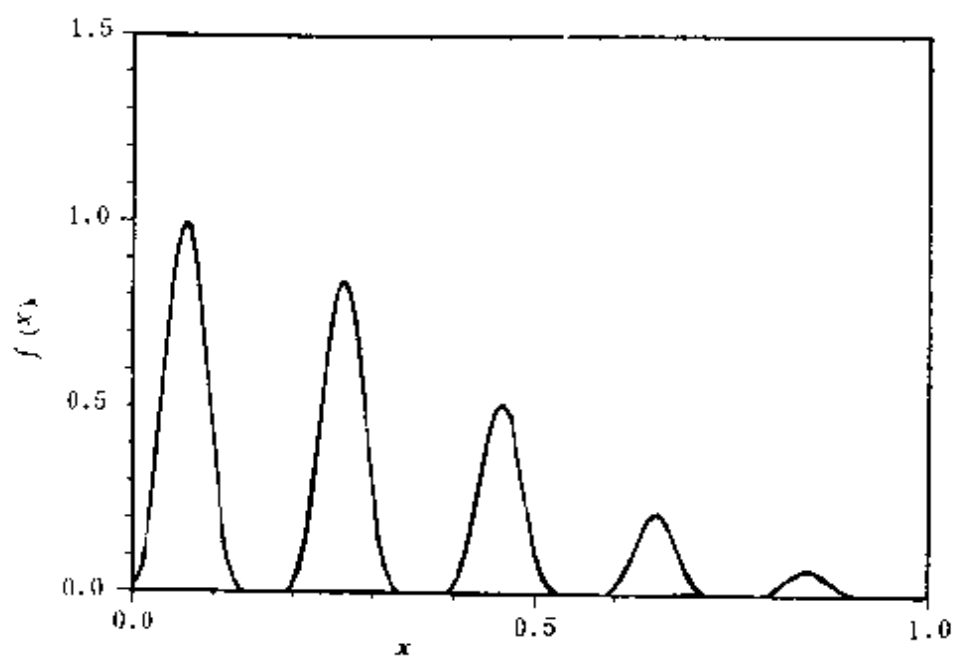


图 3.6 变峰的单变量多极值点函数

变峰的单变量多极值点函数,不难预料,基本遗传算法的优化结果将使群体集中到最高的一个山峰上。在实际应用中,我们有时需要了解问题空间内其它山峰的情况,显然,基本遗传算法不能满足这一性能要求。另外,基本遗传算法采取的是一种随机交配方式;而在生物界,交配则不完全是随机的,至少有性别、区域以及物种类别等方面因素的制约。从遗传算法角度来看,虽然随机交配方式增强了开劈新的、可能是有用的搜索空间的能力,但由于缺乏对可能的交配效果(子代的质量)方面的考虑,也会带来交配的有效性以及优化效率不太理想等方面的问题。为了解决这些问题,在遗传算法中引入小生境技术已被一些实验研究证实是种有效的尝试。

直接或间接地模拟小生境的方法已出现多种,下面介绍几种有代表性的小生境技术:

1. 基于预选择(preselection)机制的小生境技术

1970年,Cavicchio 率先在遗传算法中引入了基于预选择机制的小生境技术^[7,8]。在这种预选择机制中,只有在子串的适应度超过其父串的情况下,子串才能替换其父串,进入下一代群体。由于这种方式趋向于替换与其本身相似的个体(父与子之间的性状遗传),因而能够较好地维持群体的分布特性。Cavicchio 声称使用这种方法可以在群体规模相对较小的情况下维持较高的群体分布特性。

2. 基于排挤(crowding)机制的小生境技术

1975年,De Jong 一般化了 Cavicchio 的预选择机制,提出了一种称作排挤(crowding)机制的小生境技术^[9]。在 De Jong 的排挤机制中,他使用了群体的代间覆盖方式,依据相似性替换群体中的个体。具体算法实现步骤如下:

- (1) 初始化(建立初始群体,确定遗传参数,设定排挤因子 CF);
- (2) 计算个体的适应度;
- (3) 遗传操作(选择、交叉和变异);
- (4) 从当前群体中随机选取群体规模的 $1/CF$ 个个体组成排挤因子成员;

(5) 比较新产生的个体与排挤因子成员之间的相似性;

(6) 用新产生的个体去替换排挤因子成员中最相似的个体,形成新的当前群体;

(7) 如未满足算法终止条件则返回第(2)步,否则算法终止。

上述遗传算法在优化的初始阶段,由于群体中个体间的相似性相差不大,个体的更新替换呈随机选择特性;随着遗传优化的进展,群体中的个体逐步被分类,形成若干个小生境,此时,基于个体相似性的替换技术可在一定程度上维持群体的分布特性,并为更进一步的分类和小生境的形成腾出空间。De Jong 曾将这一技术应用到多峰函数的遗传优化上,获得了比较满意的效果。

3. 基于共享(sharing)机制的小生境技术

1987年, Glodberg 和 Richardson 提出了一种基于共享(sharing)机制的小生境技术。在这种机制中, Glogberg 和 Richardson 定义了共享函数(sharing function),用来确定每个个体在群体中的共享度。一个个体的共享度等于该个体与群体内的各个其它个体之间的共享函数值的总和。共享函数是关于两个个体之间的关系密切程度(基因型的相似性或表现型的相似性)的函数;当个体间关系比较密切时,共享函数值较大,反之,则共享函数值较小。设 d_{ij} 表示个体 i 和个体 j 之间的关系密切程度, S 为共享函数, S_i 表示个体 i 在群体中的共享度,则有:

$$S_i = \sum_{j=1}^n S(d_{ij}) \quad (3.1)$$

在计算了各个体的共享度后,个体的适应度 $f(i)$ 依据下式调整为 $f_s(i)$:

$$f_s(i) = f(i)/S_i \quad (3.2)$$

显然,这种机制限制了群体内某一特殊“物种”的无控制的增长。图3.7~图3.10显示了引入基于共享机制的小生境技术的遗传算法(GA)与基本遗传(SGA)算法在多峰函数优化方面的搜索特性差异。

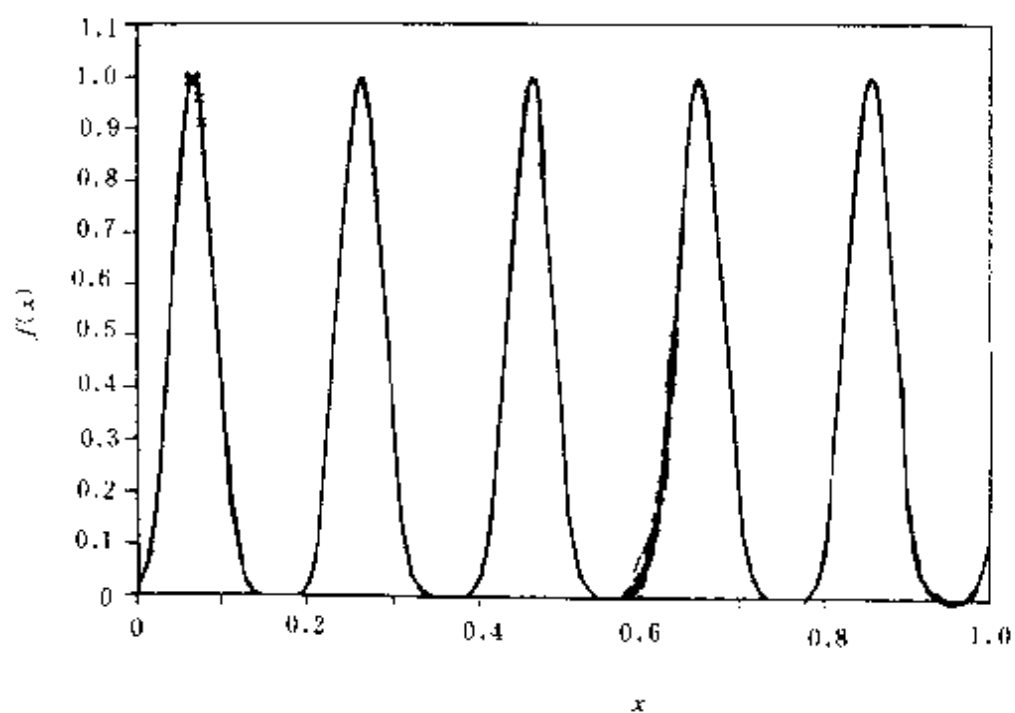


图 3.7 SGA 优化等峰单变量多极值点函数的解群分布趋势

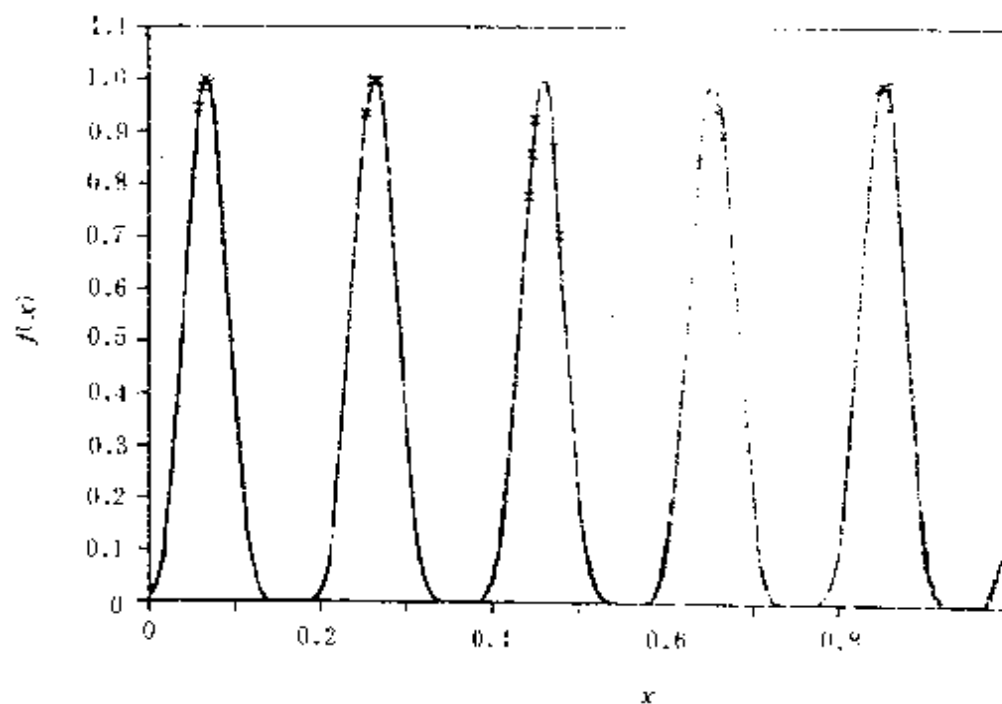


图 3.8 基于小生境技术 GA 优化等峰单变量多极值点函数的解群分布趋势

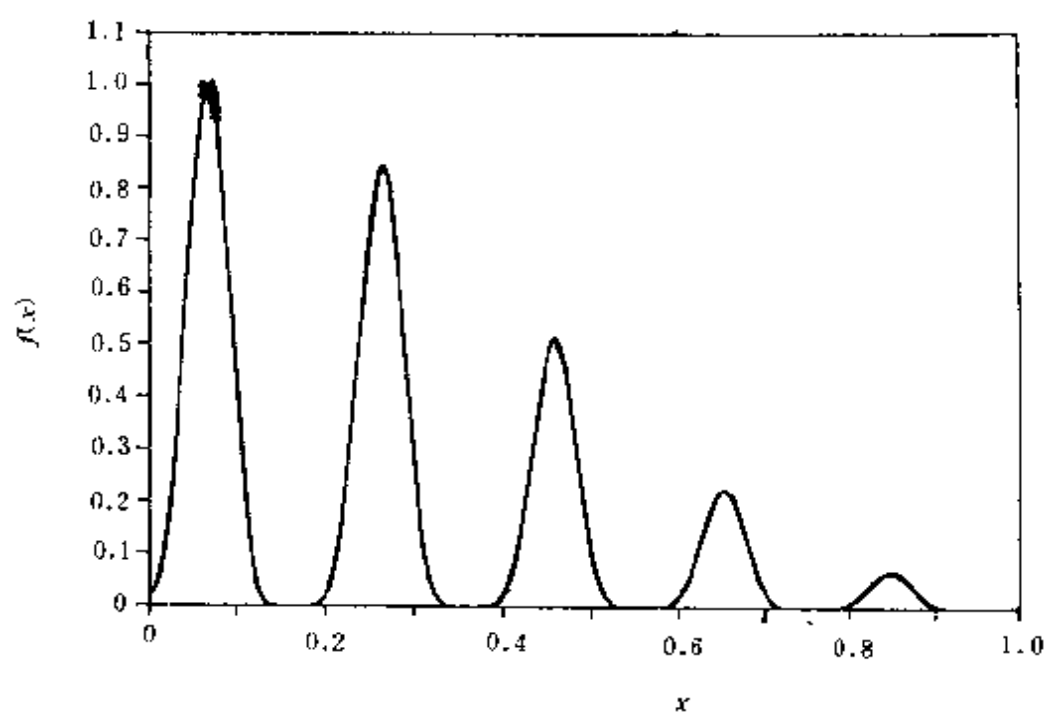


图 3.9 SGA 优化变峰单变量多极值点函数的解群分布趋势

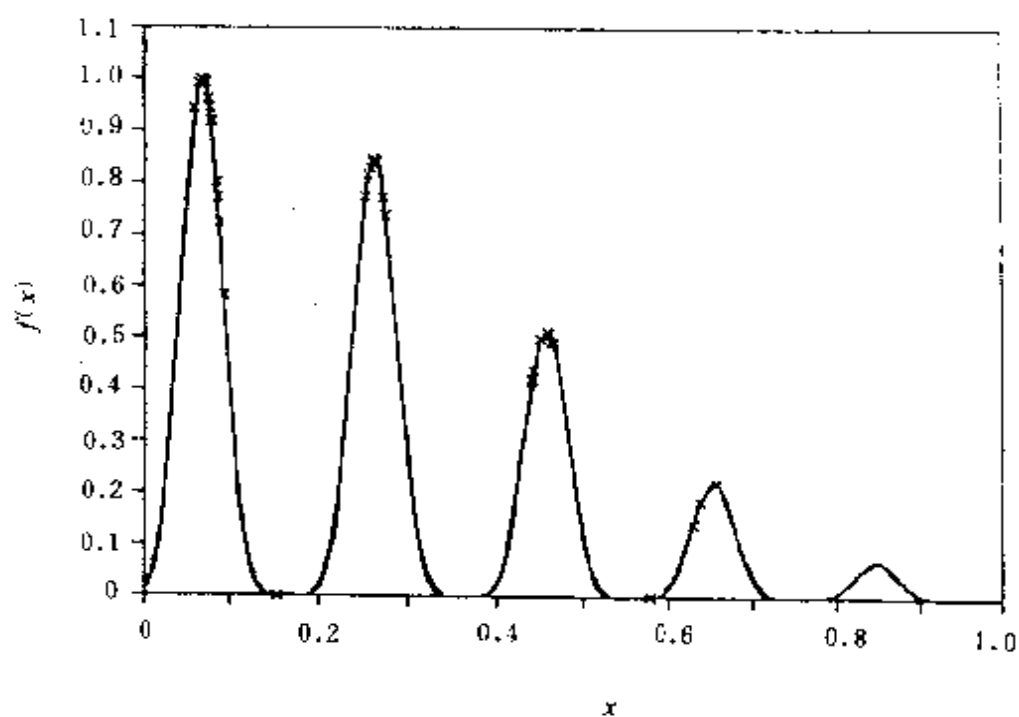


图 3.10 基于小生境技术 GA 优化变峰单变量多极值点函数的解群分布趋势

3.2 函数优化(function optimization)

广义上讲,所有的搜索、优化都是对目标函数的“函数”优化。这里所指的函数主要强调函数的数学特征,如函数的连续性、凹凸性、多峰性、多维性等。遗传算法应用于纯数学函数的优化问题,可以简化遗传算法的工作环境和优化标准,有利于分析和研究遗传算法的基本性能特征。

自1971年美国 Michigan 大学的 Hollstien 首先将遗传算法应用于纯数学问题的函数优化^[3]以来,已有多人从事这方面的研究工作。其中,De Jong 的工作最有影响;他于1975年完成了“一种遗传自适应系统的行为分析”的重要论文,其研究成果可视作遗传算法发展史上的里程碑。由于 De Jong 将 Holland 的模式理论和他自己的详细的计算实验巧妙地结合起来,对基本遗传算法及其部分变体形式的搜索性能进行了相当细致的、多层次、多方面的实验观察和分析研究,得出了一些很有价值的结论,对后来的遗传算法发展产生了十分重要的影响;因此在这里有必要较为详细地介绍一下 De Jong 在函数优化方面的研究工作,并基于其实验研究结果,对基本遗传算法及其部分变体形式的搜索性能进行若干分析和比较。

3.2.1 问题描述

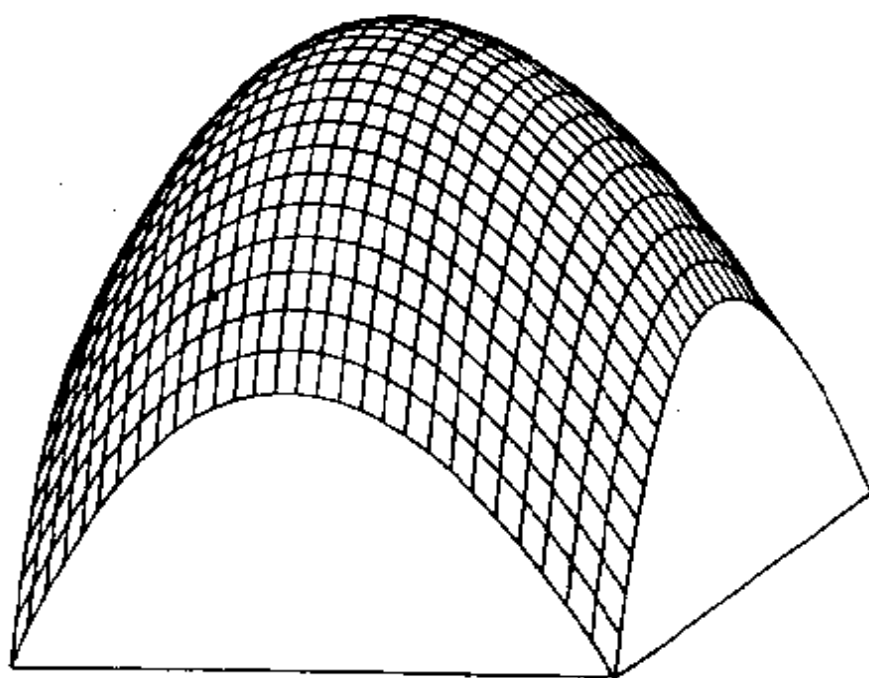
De Jong 仔细挑选的五个纯数学函数及其定义域如表3.3所示,经转化的二维函数特性分别如图3.11~3.15所示。

表 3.3 De Jong 所选的五个测试函数

| 序号 | 函 数 | 定义域 |
|----|---------------------------------|-----------------------------|
| 1 | $f_1(x_i) = \sum_{i=1}^3 x_i^2$ | $-5.12 \leq x_i \leq +5.12$ |

续表

| 序号 | 函 数 | 定义域 |
|----|--|---------------------------------|
| 2 | $f_2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$ | $-2.048 \leq x_i \leq +2.048$ |
| 3 | $f_3(x_i) = \sum_{i=1}^5 \text{integer}(x_i)$ | $-5.12 \leq x_i \leq +5.12$ |
| 4 | $f_4(x_i) = \sum_{i=1}^{30} x_i^4 + \text{Gauss}(0,1)$ | $-1.28 \leq x_i \leq +1.28$ |
| 5 | $f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^2 (x_i - a_{ij})^6}$ | $-65.536 \leq x_i \leq +65.536$ |

图 3.11 函数 $F1$ 的几何特性

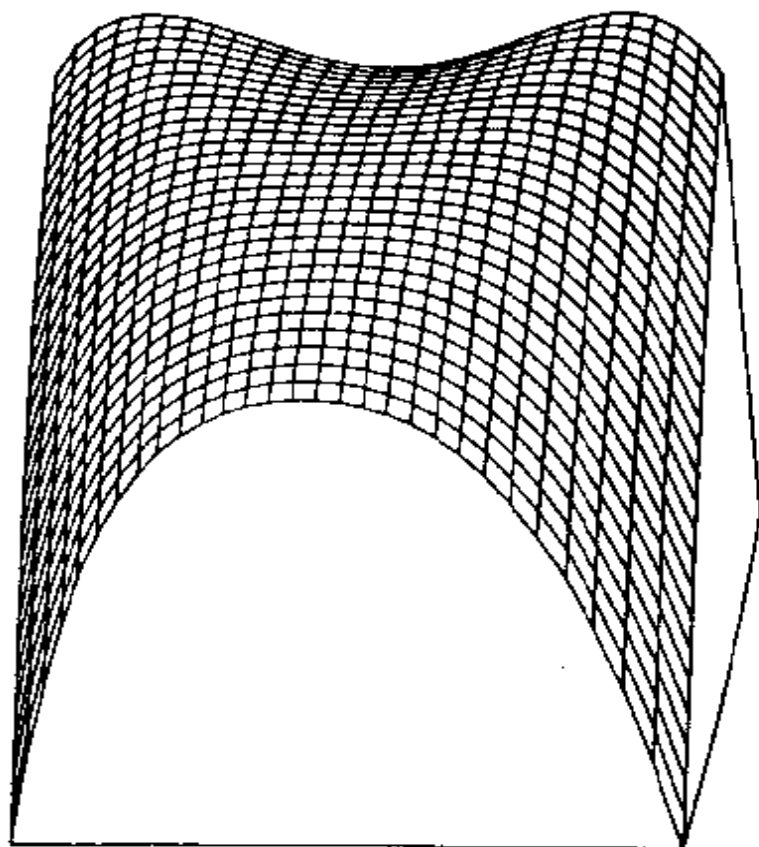


图 3.12 函数 $F2$ 的几何特性

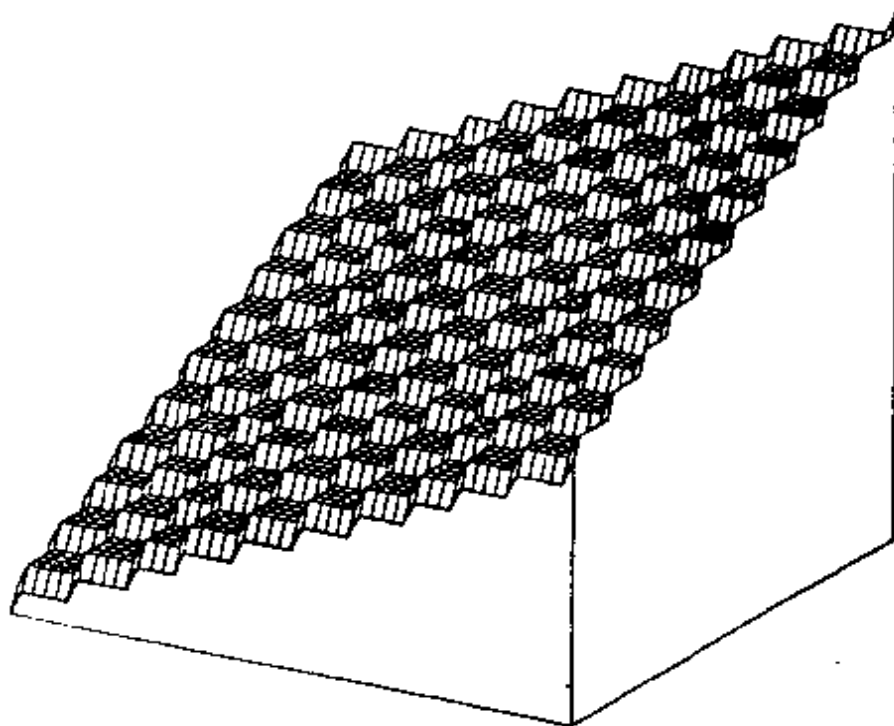


图 3.13 函数 $F3$ 的几何特性

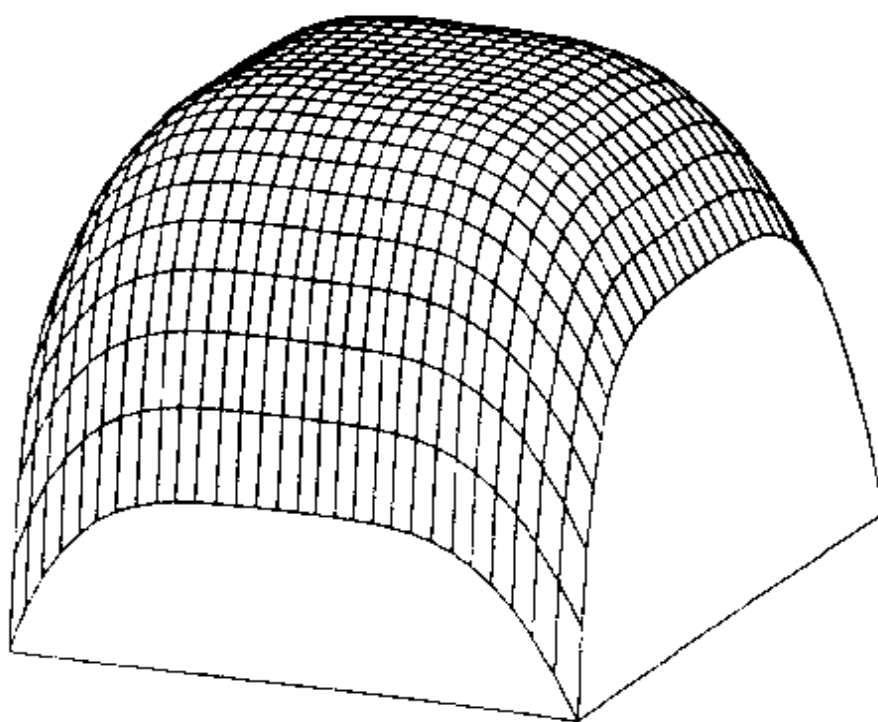


图 3.14 函数 F_4 的几何特性

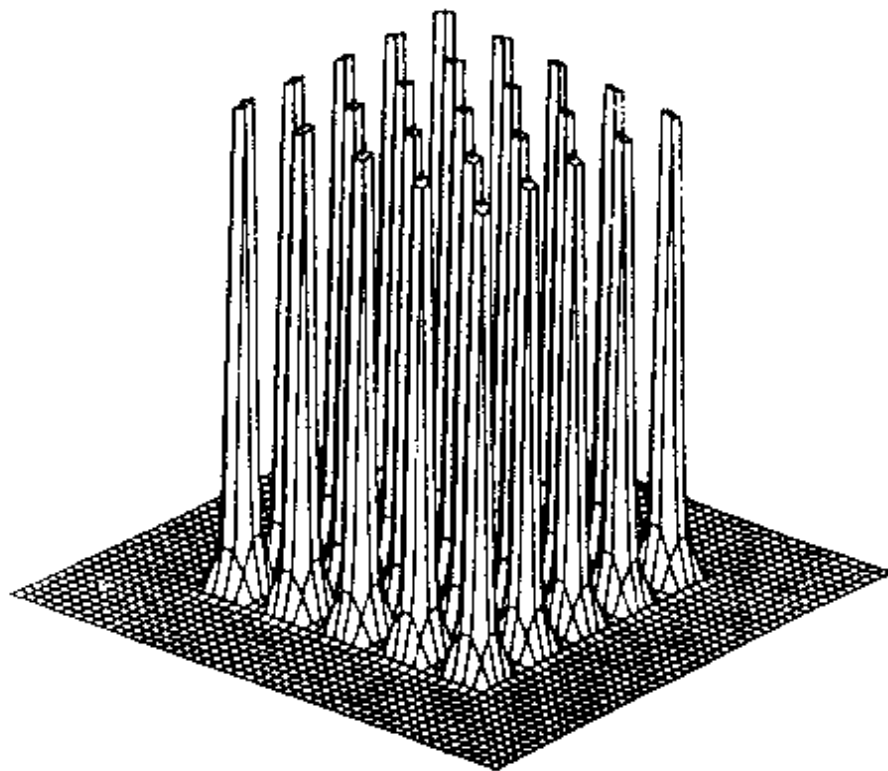


图 3.15 函数 F_5 的几何特性

表3.3 中的5个函数数学特征方面的差异主要表现在以下几个方面:

- (1) 连续性;
- (2) 凹凸性;
- (3) 二次函数与非二次函数;
- (4) 低维与高维函数;
- (5) 确定函数与随机函数;
- (6) 单峰与多峰函数;

例如,函数 $F1$ 是一个三维的、连续的、单极值(单峰)二次函数;函数 $F4$ 有随机特征,函数 $F5$ 有多峰特性等。

为了度量不同 GA 的性能,De Jong 定义了离线特性和在线特性两种测度(详见第二章第 2.5 节)。

依据这两种性能评价标准,针对由表3.3 中的5 个基本函数所构成的测试环境,De Jong 探讨和研究了基本 GA 及其变体形式的搜索优化性能。

3.2.2 编码与适应度函数

对于这种实数值的参数编码,选用二进制编码方案比较合适。码的长度可根据问题的实际需要及允许的计算资源合理地选定。例如,针对函数 $F1$,可选用30位二进制编码,每个变量10位。 $X_i = -5.12$ 对应10位二进制码串全为0 的情况, $X_i = +5.12$ 对应10位二进制码串全为1 的情况。基于相似的考虑,函数 $F2$ 的每个变量可取12位二进制编码方案,函数 $F4$ 的每个变量可取8 位编码方案,函数 $F5$ 的每个变量可取17位编码方案等。当然,也可根据计算机实现的方便选用其它的码长,如8 位、16位等。一般,设变量 X 的左边界实数值为 X_l ,右边界实数值为 X_r ,每个变量的二进制编码码长为 l ,码串对应的十进制整数值为 k ,则可依据式(3.3)进行参数解码。

$$X = (X_r - X_l) \cdot k/2^l + X_l \quad (3.3)$$

由于五个函数的值域均为正的实数域,优化的目标是寻求最大

值,故可直接取各函数的目标函数值作为适应度(在 De Jong 的研究中,未使用适应度函数的定标技术)。

3.2.3 基本遗传算法(SGA)的搜索性能

为了后面的叙述方便,这里我们把基本遗传算法(SGA)记作策略 $R1$, $R1$ 限定为:

- (1) 赌轮选择机制;
- (2) 一点交叉(随机交配);
- (3) 位点变异;
- (4) 所有操作使用二进制位串群体。

De Jong 的 $R1$ 策略考虑了下述相互制约的4个参数:

- (1) 群体规模 n ;
- (2) 交叉概率 P_c ;
- (3) 变异概率 P_m ;
- (4) 代沟(gap) G , 其中 $G=1$ 表示非覆盖群体; $0 < G < 1$ 表示覆盖群体。

在覆盖群体的情况下, $n \cdot G$ 个个体被选择用于更进一步的遗传算法操作。产生的子串放到 $n \cdot G$ 群体槽(均匀随机地,不替换)。

在 De Jong 的研究中,他先考察了策略 $R1$ 时单个参数发生变化的情况,继而考察了它们的有限的组合情况。图3.16~3.19显示了关于函数 $F1$ 的群体规模实验结果,正如所期望的,较大规模的群体能导致较好的离线性能(收敛性),由于大群体的分布模式的有效性,较大群体也显示了初始在线特性较差的结果。另一方面,较小群体显示了更快的搜索能力,因而展示了较好的初始在线特性。

为了对付不成熟(premature)收敛情况下的重要基因损失,增加变异速率是个办法,这种方法可维持足够的分布。在继续改进过程中,De Jong 的研究已经清楚地证明这种办法也不是万能的,图3.19~3.21显示了这一特征。在这些运行中($n=50, P_c=1.0, G=1.0$),显然增加 P_m ,减少了基因损失数目,但这种减少是由降低离线和在

线性能为代价的。当 P_m 上升到 0.1 时, $R1$ 的离线性能越来越像一种单纯随机搜索的情况(当然, $P_m = 0.5$ 是一种随机搜索, 不考虑 P_c 和 n 值), 而且, P_m 增加还降低了在线性能。

De Jong 也考察了交叉概率和代沟值, 作为结果, 他建议 $P_c = 0.6$ 可作为在离线和在线性能之间的权衡, 后来的研究^[10] 建议 $P_c = 1.0$ 。关于代沟研究, 他建议非覆盖的模型是最好的(在大多数优化研究中), 离线性能较优。不过, De Jong 的研究也确实展示当使用较小的代沟值时, 在线特性并不严格减少。这个事实对于机器学习特别有用, 因为在机器学习问题中, 边学习边执行是十分重要的。

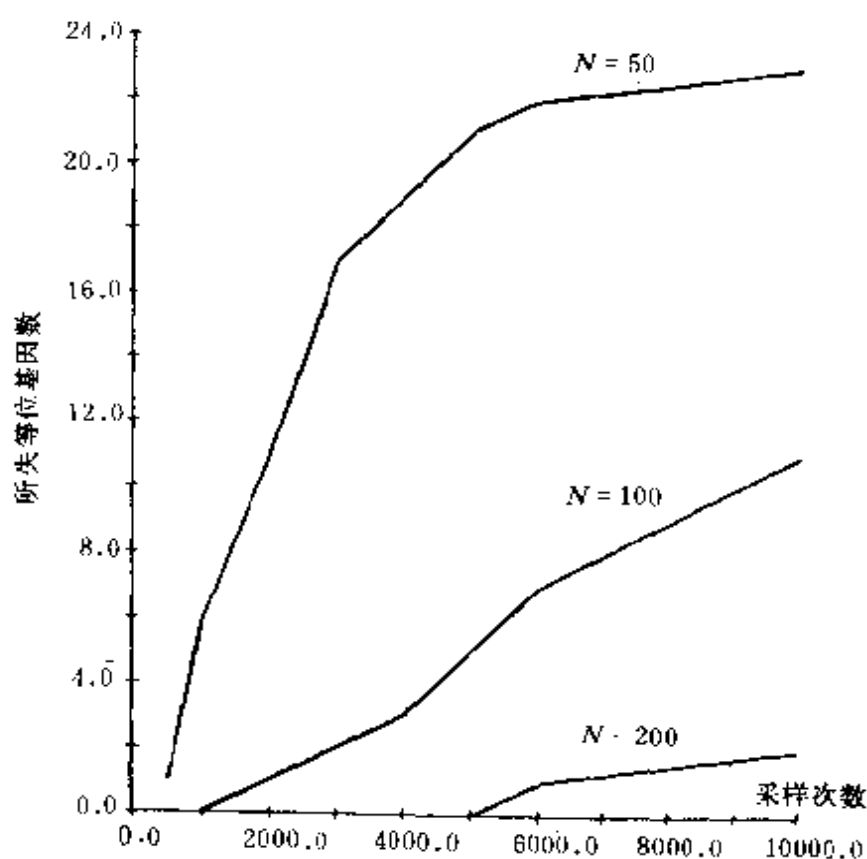


图 3.16 群体规模对等位基因损失的影响(优化策略为 $R1$, 测试函数为 $F1$)

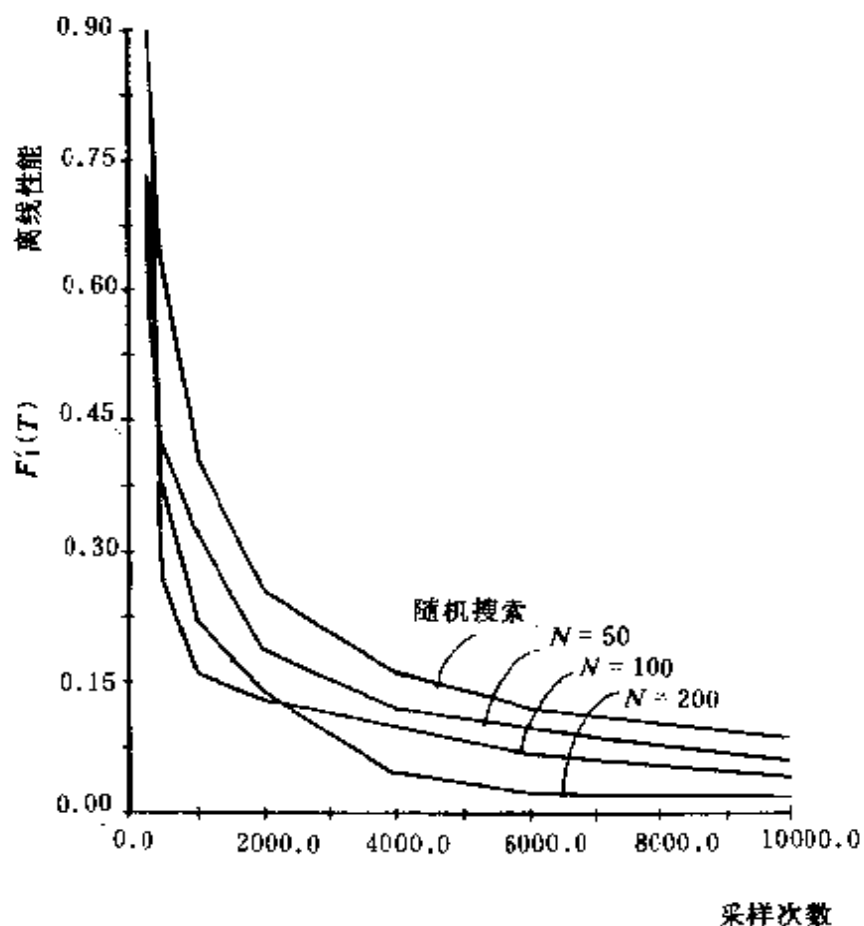


图 3.17 群体规模对离线性能的影响(优化策略为 R1, 测试函数为 F1)

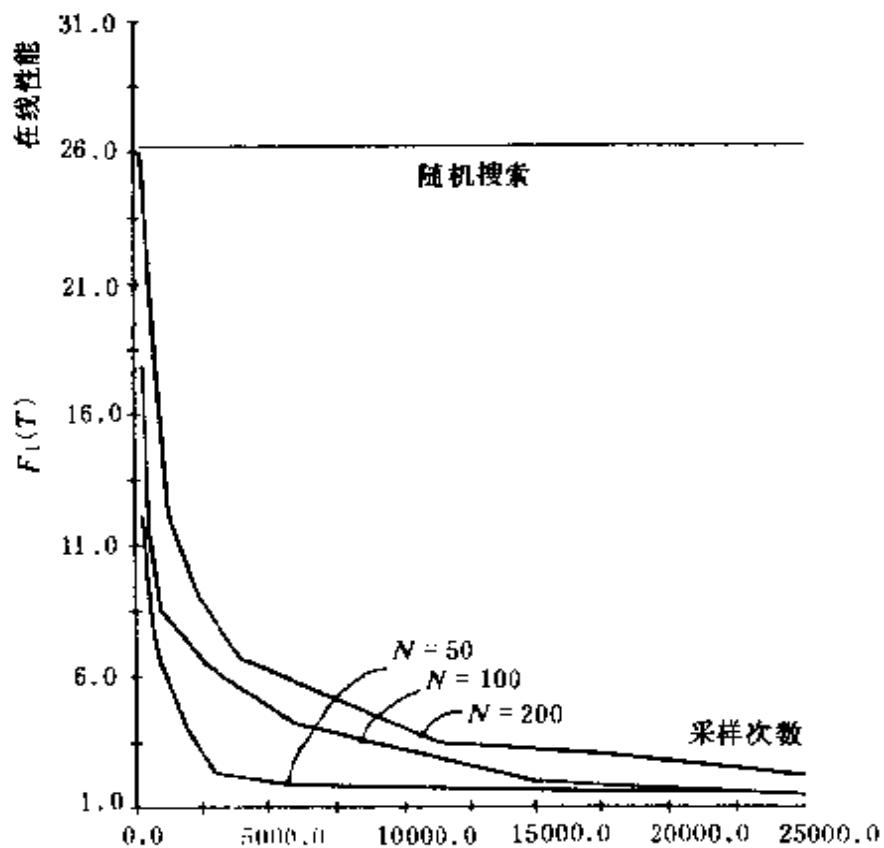


图 3.18 群体规模对在线性能的影响(优化策略为 R1, 测试函数为 F1)

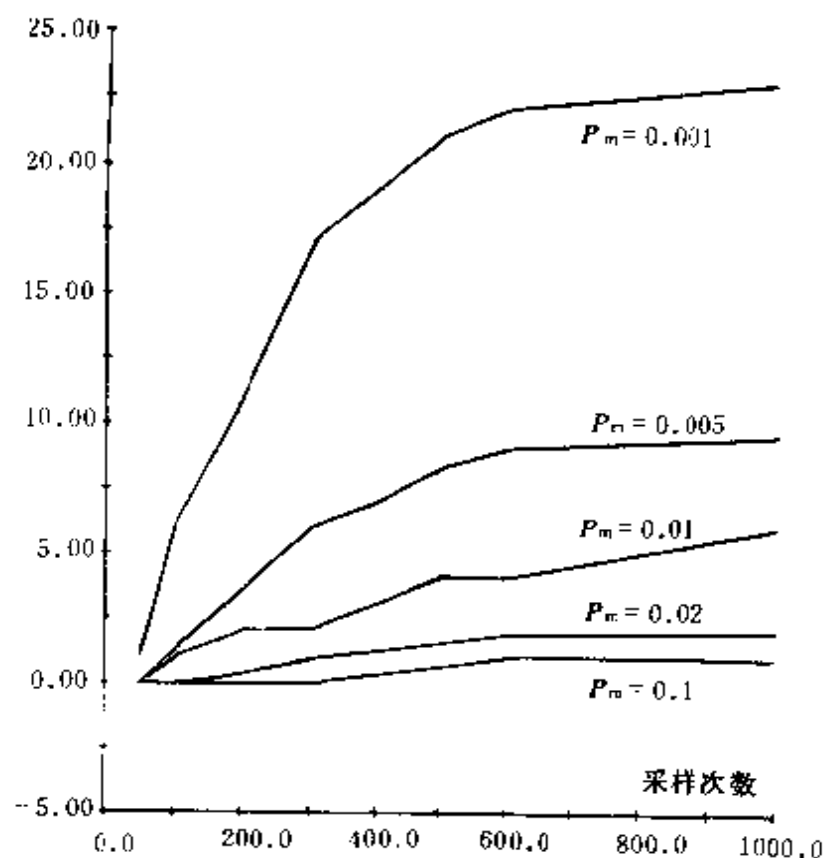


图 3-10 变异概率对等位基因损失的影响(优化策略为 R1, 测试函数为 F1)

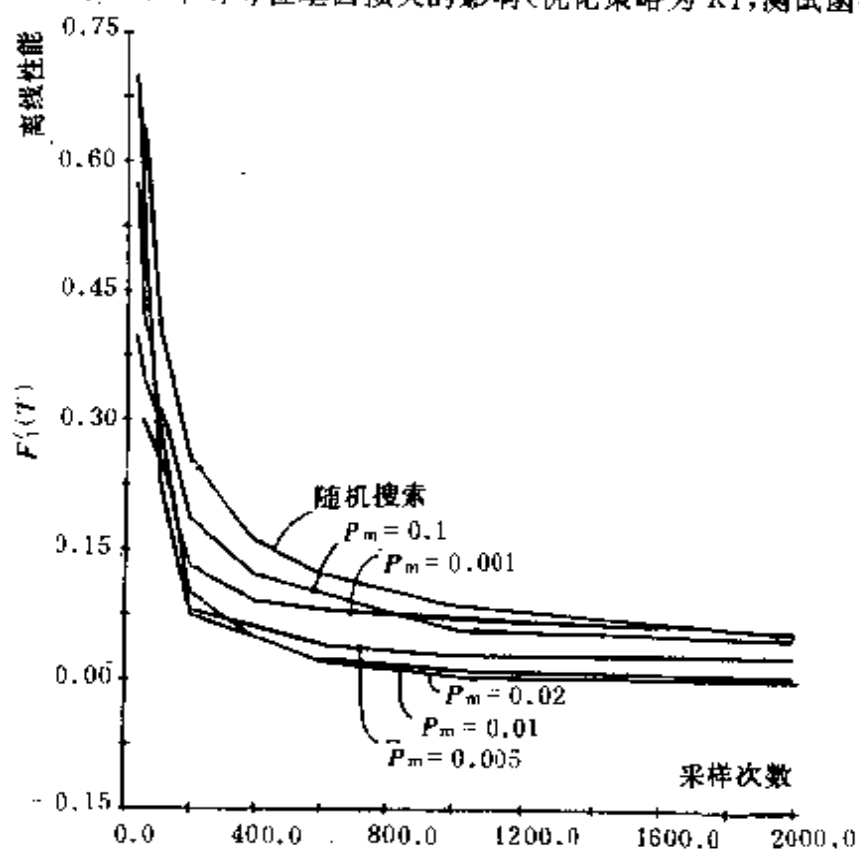


图 3-11 变异概率对等位基因损失的影响(优化策略为 R1, 测试函数为 F1)

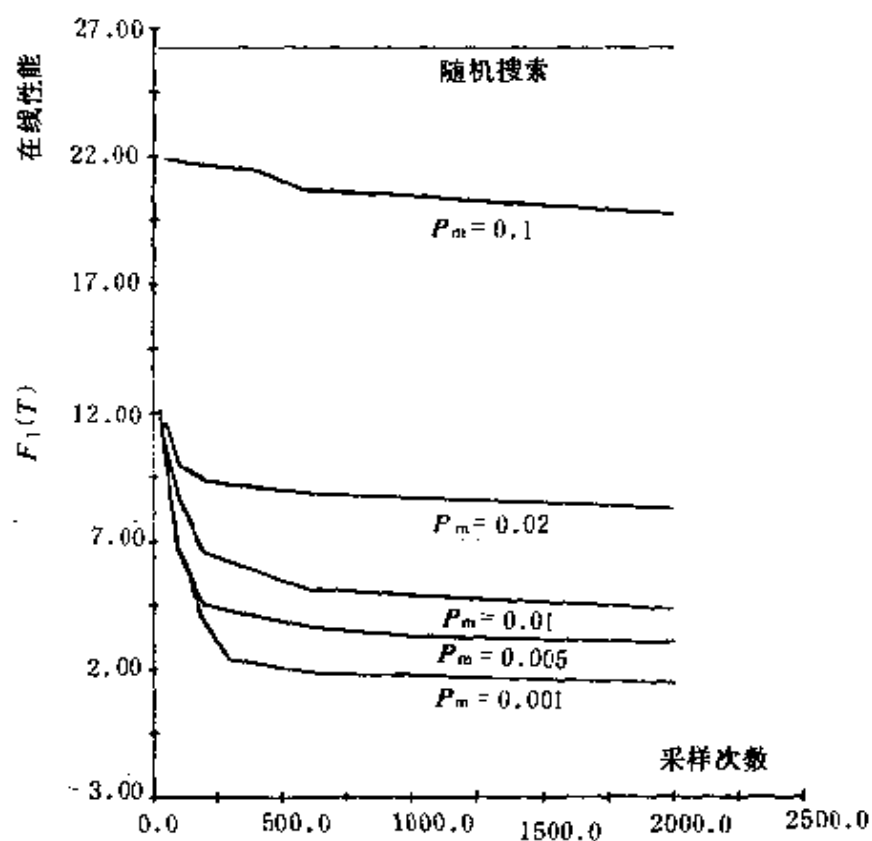


图 3.19 变异概率对在线性能的影响(优化策略为 R_1 , 测试函数为 F_1)

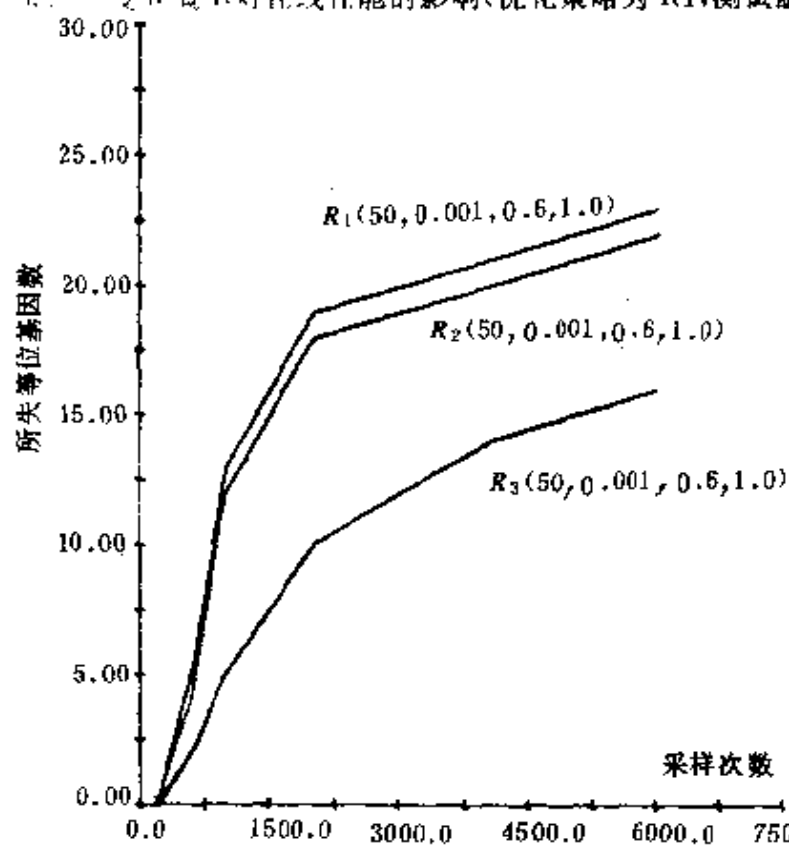


图 3.22 优化策略 R_1 , R_2 与 R_3 在基因丢失方面的性能比较(测试函数为 F_1)

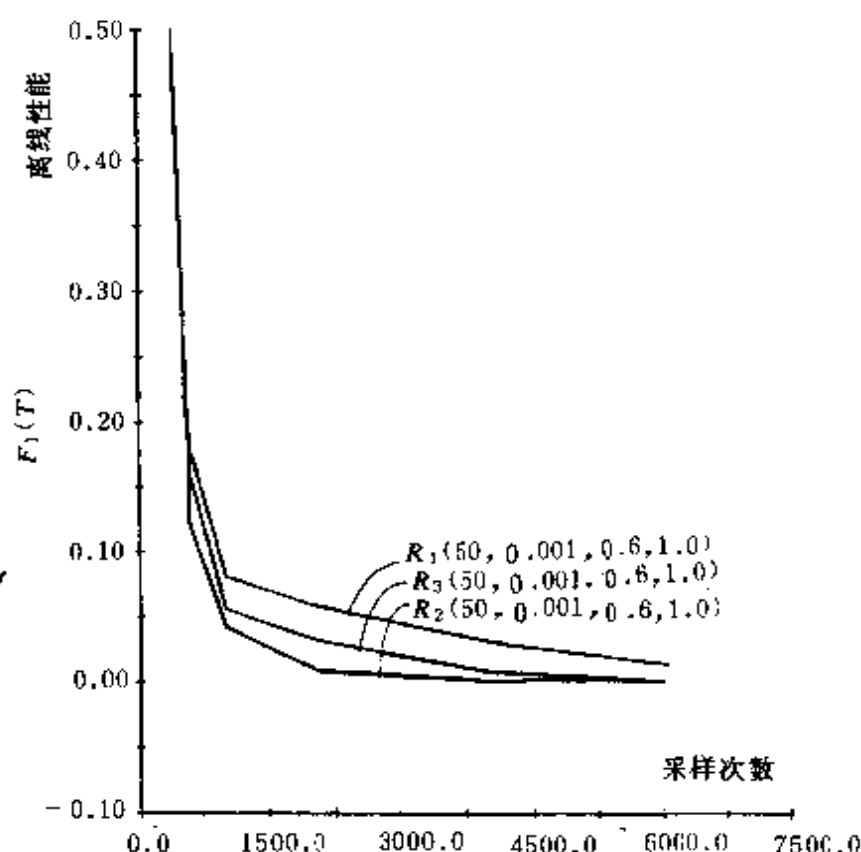


图 3.23 优化策略 R_1 , R_2 和 R_3 的离线性能比较(测试函数为 F_1)

3.2.4 基本遗传算法的若干变体形式的搜索性能

为了改进基本遗传算法的性能, De Jong 探索了策略 R_1 的五种如下变形:

R_2 : 最佳保留模型(elitist model);

R_3 : 期望值模型(expected value model);

R_4 : 最佳保留期望值模型(elitist expected value model);

R_5 : 排挤因素模型(crowding factor model);

R_6 : 一般交叉模型(generalized crossover model)。

在 R_2 实验中, De Jong 发现对于单峰函数, 最佳保留策略大大改进了在线和离线性能。不过, 在多峰函数 F_5 中, 最佳保留策略降低了两种性能, 正如 De Jong 指出的, 这也许意味着最佳保留策略改进了局部搜索性能, 却损失了全局搜索特性。

为了减少赌轮选择的随机错误,De Jong 设计了期望值模型 R_3 ,图3.22比较了期望值策略 R_3 和策略 R_2 及策略 R_1 在优化函数 F_1 时群体的等位基因损失情况。 R_3 在基因损失方面优于 R_2 及 R_1 。图3.23、3.24比较了 R_3 与 R_1, R_2 的离线和在线性能, R_3 明显在离线和在线性能上优于 R_1 ,而且,虽然 R_2 在低维、单峰的函数 F_1 中其在线和离线性能略优于 R_3 ,但 R_3 在5个函数的环境中其整体搜索性能优于 R_2 和 R_1 。De Jong 还发现 R_3 减少的随机错误能够容许交叉概率的适度增加。Booker^[11], Brindle, Grefenstette^[16]在关于减少随机错误的采样程序研究中确认了这一观察。

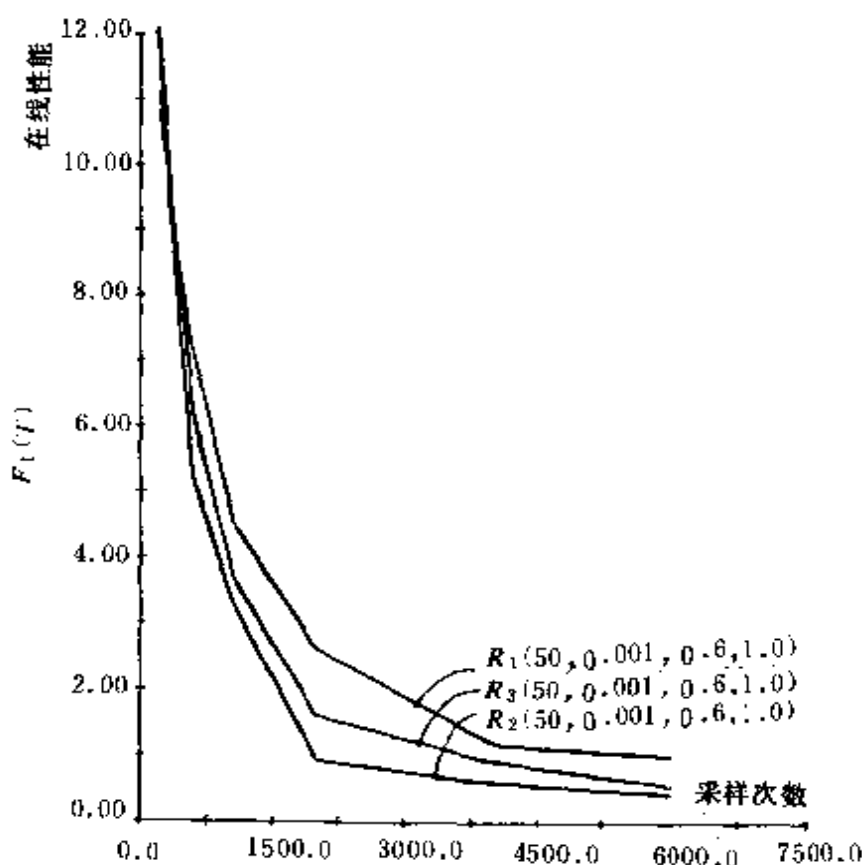


图 3.24 优化策略 R_1, R_2 和 R_3 的在线性能比较(测试函数为 F_1)

在策略 R_4 中,正如人们所期望的,对于 $F_1 \sim F_4$ 单峰函数可观察到相当可观的改进。而对于多峰函数 F_5 ,其特性仅低于 R_3 。不过,由于对前面的4个函数的优化性能改进,因此,其整体鲁棒性是稍胜一筹的。

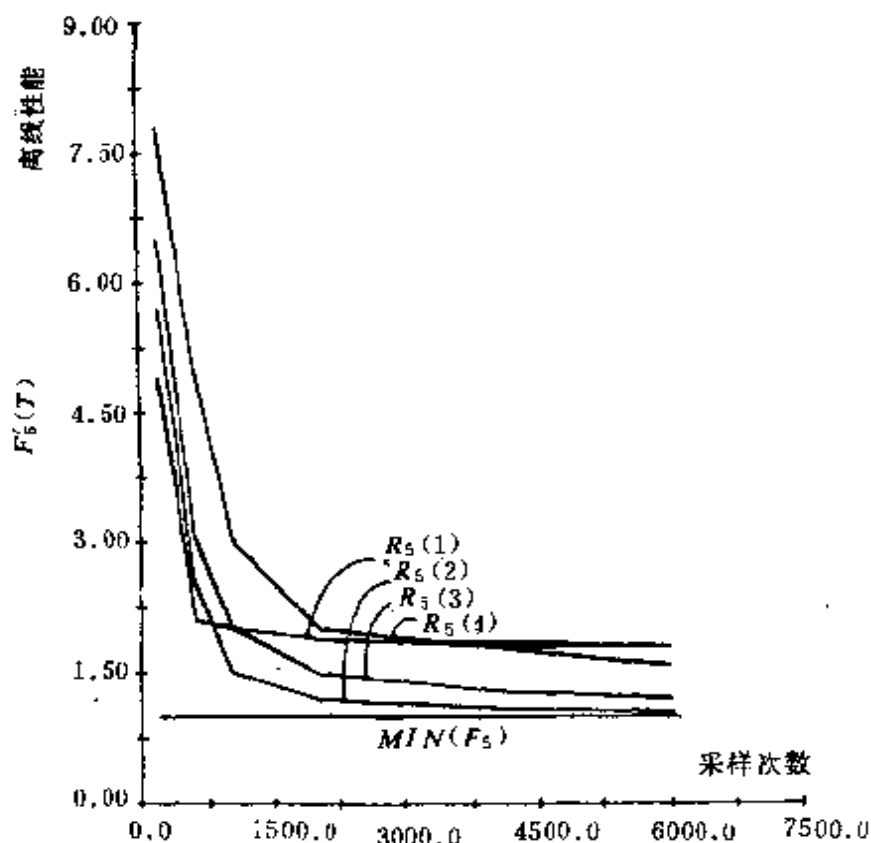


图 3.25 排挤因子对离线性能的影响(优化策略为 R5, 测试函数为 F5)

基于 $R4$ 在函数 $F5$ 优化上的性能降低, De Jong 随后设计了 $R5$ 。跟 Holland 一样, De Jong 认为, 在自然界, 当相似的个体开始确定一个小生境(niche)时, 由于资源限制, 增加的竞争减少了生命期望值及出生率。在小生境不太密集, 低选择压力时, 生命期望和出生率大多与他们潜能相近。为了在人工遗传算法中加入如此的排挤压力, De Jong 将新近产生的子串强制替换相似的、较老的成熟串, 以期维持群体中更大的分布, 即多样性。

为了这样做, De Jong 采用了一个覆盖群体模型且使用代沟值 $G=0.1$, 他也定义了一个新的称做 CF 的参数。在 $R5$ 中, 排挤模型使得一个个体诞生时, 便选择一个个体死亡。死亡个体为随机选取的 n/CF 个个体中与新生个体最相似的个体。

图 3.25 表明, 当 $CF=2$ 时, 给出了一个逼近全局最优的性能 ($F5$), 排挤和小生境思想已由其它人进一步开拓 (Goldberg &

Richardson, 1987; Perry, 1984)。这些思想对于多峰函数优化及机器学习是十分重要的。

De Jong 考虑的最后一种模型是一般化交叉模型 R6。在 R6 中, 要定义参数 CP(交叉点数)。当 $CP=1$ 时为 SGA。当 CP 取偶数值时, 串被处理成无始无终的圆圈; 当 $CP=3$ 时, 0 位点是一个默认的交叉点。

在 De Jong 的研究中, 得出的结论是多点交叉随交叉点的增长降低了在线和离线性能。当 CP 为多点时, 若串长为 L, 则有 (LCP) 种交叉操作。这一数目的增加将极大可能破坏重要的模式。

一般来说, 使用遗传算法求解光滑的、单峰函数是没有什么意义的, 但如果遗传算法能在鲁棒性方面达到一定的期望值, 则遗传算法就提供了一种令人鼓舞的、有前途的优化方法。

3.3 背包问题(knapsack problem)

背包问题是一个典型的组合优化问题。本节介绍基于遗传算法优化背包问题的具体方法。

3.3.1 问题描述

0/1 背包问题: 给出几个尺寸为 S_1, S_2, \dots, S_n 的物体和容量为 C 的背包, 此处 S_1, S_2, \dots, S_n 和 C 都是正整数; 要求找出 n 个物件的一个子集使其尽可能多地填满容量为 C 的背包。用数学形式可以更精确地描述如下:

$$\begin{aligned} & \text{最大化} && \sum_{i=1}^n S_i X_i \\ & \text{满足} && \sum_{i=1}^n S_i X_i \leq C \\ & && X_i \in \{0, 1\} \quad 1 \leq i \leq n \end{aligned} \quad (3.4)$$

还有一种更为广义的背包问题, 它比上面的描述更为有用, 其输

入由 C 和两个向量 $C=(S_1, S_2, \dots, S_n)$ 和 $P=(P_1, P_2, \dots, P_n)$ 组成。设 X 为一整数集合, 即 $X=1, 2, 3, \dots, n$, T 为 X 的子集, 则问题就是找出满足约束条件 $\sum_{i \in T} S_i \leq C$, 而使 $\sum_{i \in T} P_i$ 获得最大的子集 T , 即求 S_i 和 P_i 的下标子集。在应用问题中, 设 S 的元素是 n 项经营活动各自所需的资源消耗, C 是所能提供的资源总量, P 的元素是人们从每项经营活动中得到的利润或收益, 则背包问题就是在资源有限的条件下, 追求总的最大收益的资源有效分配问题。广义背包问题可以数学形式更精确地描述如下:

$$\begin{aligned} & \text{最大化} \quad \sum_{i=1}^n P_i X_i \\ & \text{满足} \quad \sum_{i=1}^n S_i X_i \leq C \\ & \quad \quad X_i \in \{0, 1\} \quad 1 \leq i \leq n \end{aligned} \quad (3.5)$$

背包问题在计算理论中属于 NP ——完全问题, 其计算复杂度为 $O(2^n)$; 若允许物件可以部分地装入背包, 即允许 X_i 可取从 0.00 到 1.00 闭区间上的实数, 则背包问题就简化为极简单的 P 类问题, 此时计算复杂度为 $O(n)$ 。

3.3.2 遗传编码

采用下标子集 T 的二进制编码方案是常用的遗传编码方法。串 T 的长度等于 n (问题规模), $T_i (1 \leq i \leq n) = 1$ 表示该物件装入背包, $T_i = 0$ 表示不装入背包。基于背包问题有近似求解知识, 以及考虑到遗传算法的特点 (适合短定义长的、低阶的、高适应度的模式构成的积木块结构类问题), 通常将 P_i, S_i 按 P_i/S_i 值的大小依次排列, 即 $P_1/S_1 \geq P_2/S_2 \geq \dots \geq P_n/S_n$ 。

3.3.3 适应度函数

在上述编码情况下, 背包问题的目标函数和约束条件可表示如下。

$$\text{目标函数: } J(T) = \sum_{i=1}^n T_i P_i$$

$$\text{约束条件: } \sum_{i=1}^n T_i S_i \leq C$$

按照利用惩罚函数处理约束条件的方法,我们可构造背包问题的适应度函数 $f(T)$ 如下式:

$$f(T) = J(T) + g(T) \quad (3.6)$$

式中 $g(T)$ 为对 T 超越约束条件的惩罚函数,惩罚函数可构造如下:

$$g(T) = \begin{cases} 0, & C - \sum_{i=1}^n T_i S_i \geq 0 \\ \beta E_m (C - \sum_{i=1}^n T_i S_i), & C - \sum_{i=1}^n T_i S_i < 0 \end{cases} \quad (3.7)$$

式中 E_m 为 $P_i/S_i (1 \leq i \leq n)$ 的最大值, β 为合适的惩罚系数。

3.3.4 基于基本遗传算法求解背包问题

根据背包问题的性质及上述遗传编码的特点,选用基本遗传算法(SGA)求解背包问题是相当自然的。基本遗传算法由下述基本三算子构成。

- (1) 正比于适应度的赌轮随机选择方式;
- (2) 一点交叉,交叉概率为 p_c ;
- (3) 位点变异,变异概率为 P_m 。

下面就背包问题的遗传优化举两个实例:

例3.1 背包问题实例1

假设 $S = \{255, 234, 199, 182, 180, 169, 144, 111, 105, 88, 77, 68, 58, 55, 49, 37, 26, 22, 19, 11\}$;

$C = 1882.67$;

$X_i \in [0, 1], 1 \leq i \leq n$; (与式(3.4)有点差异,此处 X_i 取实数)

求使 $\sum_{i=1}^n S_i X_i$ 最大且满足约束条件:

$$\sum_{i=1}^n S_i X_i \leq C \quad 0.0 \leq X_i \leq 1.0, \quad 1 \leq i \leq n$$

的解矢量 X 。

根据背包问题的理论知识,不难证明例1 属于 P 类问题,其算法复杂度为 $O(n)$,若将 S_i 按从大到小依次排列,则其最优解为:

当 $\sum_{i=1}^j S_i X_i < C$ 时, $X_i = 1, 1 \leq i \leq j$

$$X_{j+1} = (C - \sum_{i=1}^j S_i) / S_{j+1}$$

$$X_i = 0, j+2 \leq i \leq n$$

对于例1,可容易算出最优解为:

$$X = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.23, 0, 0, 0, 0, 0, 0\}.$$

若用遗传算法优化例1,可对 X_i 进行8位量化,则可用 $8 \times 20 = 160$ 位二进制码对矢量 X 编码,取惩罚系数 $\alpha = 2.01$,交叉概率 $P_c = 0.88$,变异概率 $P_m = 0.001$,取群体规模 $popsiz = 80$,选用正比于适应度的比例选择机制,随机选点的一点交叉策略,位点随机变异技术,经过200 代的遗传算法运算,我们也可得出这一最优结果。

显然,从这一实例可以看出,对于简单的计算问题,遗传算法在时空效率方面难以令人满意;遗传算法的性能也不能与基于问题知识的专用算法匹敌。但正是由于遗传算法优化方法不太需要基于问题的知识,因而在通用性、鲁棒性及复杂问题的求解方面显示了遗传算法的特色。对于实例3.1,由于遗传算法对基于背包问题特殊性的专门知识的茫然,因此,遗传算法仍就在没有约束限定的 2^{160} 的复杂空间内执行搜索,即对遗传算法来说,例3.1 仍然是一个比较复杂的问题。

例3.2 背包问题实例2

设 $S = \{253, 245, 243, 239, 239, 239, 238, 238, 237, 232,$

231, 231, 230, 229, 228, 227, 224, 217, 213, 207,
 203, 201, 195, 194, 191, 187, 187, 177, 175, 171,
 169, 168, 166, 164, 161, 160, 158, 150, 149, 147,
 141, 140, 139, 136, 135, 132, 128, 126, 122, 120,
 119, 116, 116, 114, 111, 110, 105, 105, 104, 103,
 93, 92, 90, 79, 78, 77, 76, 76, 75, 73,
 62, 62, 61, 60, 60, 59, 57, 56, 53, 53,
 51, 50, 44, 44, 42, 42, 38, 36, 34, 28,
 27, 24, 22, 18, 12, 10, 7, 4, 4, 1 ;

$C = 6666;$

$X_i \in \{0, 1\}, 1 \leq i \leq n$

求使 $\sum_{i=1}^n S_i X_i$ 最大且满足约束条件:

$$\sum_{i=1}^n S_i X_i \leq C \quad X_i \in \{0, 1\}, 1 \leq i \leq n$$

的解矢量 X 。

例3.3 是一个0/1 背包问题,属于典型的 NP 类问题。对于这一问题,我们可以直接取二进制 n 维解矢量 X 作为解空间参数的遗传编码; X 的元素为1 表示该元素所对应的物件被选中装入背包, X 的元素为0 表示该元素所对应的物件没有被选中。例如 $X = \{0, 1, 0, 1, 0, 0, 1\}$ 表示第2, 4, 7 这三个物件被选中装入背包。考虑到例2 的约束条件,可构造下述适应度函数:

$$f(X) = \begin{cases} \sum_{i=1}^n S_i X_i, & \sum_{i=1}^n S_i X_i \leq C \\ \sum_{i=1}^n S_i X_i - \alpha (\sum_{i=1}^n S_i X_i - C), & \sum_{i=1}^n S_i X_i > C \end{cases} \quad (3.8)$$

上式中, $\sum_{i=1}^n S_i X_i$ 为背包问题的目标函数值, α 为惩罚系数。

我们采用遗传算法对例3.2 进行了编程实现。实验中,取惩罚系数 $\alpha = 1.10$,交叉概率 $P_c = 0.88$,变异概率 $P_m = 0.0088$,群体规模

取 $popsiz=100$, 选用正比于适应度的比例选择机制, 随机选点的一点交叉策略, 位点随机变异技术, 初始集团以随机方式产生, 在 $gen=14$ 代时, 得到最优结果:

$$X = \{1, 1, 0, 0, 1, 1, 1, 1, 1, 0, \\ 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, \\ 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, \\ 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, \\ 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, \\ 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, \\ 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, \\ 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, \\ 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, \\ 1, 0, 1, 0, 0, 1, 0, 0, 0, 1\}$$

$$\sum_{i=1}^n S_i X_i = 6666$$

在386微机上的执行时间为123s。在实验中, 我们也看到, 当0/1背包问题的规模较大, 而背包容量 C 相对较小时, 按随机方式产生的初始群体有可能出现个体适应度全为0的情况, 而使遗传操作无所适从。因此, 在初始群体的产生算法中引入适当的“选种”机制, 以避免个体适应度全部或大部分为0的情况。

3.3.5 基本遗传算法的搜索能力

在NP-完全类组合优化问题(如作业调度问题、TSP问题等)的搜索优化应用中, 遗传算法在求解背包问题上显示了超出想象的、良好的搜索性能。小长谷、丸山等人^[22]采用如3.3.4小节所述的基本GA优化方案, 仅用0.6s就搜索到了50个物件背包问题的全局最优解(在R3000/33MHz机器上), 而采用穷举法全局搜索这一问题却需要21120.5s, 两者的搜索效率相差约81232倍。我们在386/33MHz微机上对100个物件的0/1背包问题进行的实验也显示了同样良好

的性能。这也许是由于背包问题相对来说比较简单的缘故,或者是由
于背包问题的结构特征比较适合遗传算法优化的缘由。

3.3.6 基于“与/或”交叉方法求解背包问题

我们在用遗传算法求解背包问题的实验研究中,尝试了一种“与
/或”交叉方法,它在背包问题的实验研究中显示了相对于“一点交
叉”策略来说更加优越的性能。其具体实现方法如下:

- (1) 按赌轮选择机制选取两个父串 $F1$ 和 $F2$ 。
- (2) 由 $F1$ 和 $F2$ 的按位“与”逻辑运算产生一子串 $C1$ 。
- (3) 由 $F1$ 和 $F2$ 的按位“或”逻辑运算产生另一子串 $C2$ 。

例如,选择的两父串为

$F1: 0100101101$

$F2: 1101110100$

则由“与/或”交叉方法产生的两子串分别为

$C1: 0100100100$

$C2: 1101111101$

显然,这一交叉方法使子代继承了双亲的同型基因。对于双亲的
杂型基因,“与/或”交叉方法采取了两种不同的“支配”方式:“与”运
算是一种0 支配1 的方式,而“或”运算是一种1 支配0 的“支配”方
式。

在背包问题的优化实验中,与“一点交叉”策略相比,“与/或”交
叉策略能使优化过程更加迅速地到达全局次优解,获取全局最优解
的所需计算时间仅为“一点交叉”策略的1/9 到1/3。用在线性能和
离线性能进行评估,则“与/或”策略在“在线性能”和“离线性能”两个
性能指标上均明显优于“一点交叉”策略。当然,这一交叉方法还有待
理论上、实践上进一步的研究和探索。不过,如果进一步的实际应用
能证实该策略的有效性,则“与/或”交叉策略将为遗传算法的硬件实
现创造了良好的条件。

3.4 货郎担问题

近几年来,基于遗传算法方法求解货郎担(TSP)问题的研究相当活跃。例如 David B. Fogel^[12],Grefenstette^[13],Goldberg^[14],等。在遗传算法研究中,TSP 问题已被广泛地用于评价不同的遗传操作及选择机制的性能。之所以如此,主要有以下几个方面的原因:

(1) TSP 问题是一个典型的、易于描述却难以处理的 NP 完全问题。有效地解决 TSP 问题在可计算理论上有着重要的理论价值。

(2) TSP 问题是诸多领域内出现的多种复杂问题的集中概括和简化形式。因此,快速、有效地解决 TSP 问题有着极高的实际应用价值;

(3) TSP 问题因其典型性已成为各种启发式的搜索、优化算法的间接比较标准(如神经网络优化法、列表寻优(TABU)法、模拟退火法等),而遗传算法方法就其本质来说,主要是处理复杂问题的一种鲁棒性强的启发式随机搜索算法。因此遗传算法在 TSP 问题求解方面的应用研究,对于构造合适的遗传算法框架、建立有效的遗传操作以及有效地解决 TSP 问题等有着多方面的重要意义。

TSP 问题描述十分简单,简言之就是寻找一条最短的遍历 n 个城市的路径,或者说搜索整数子集 $X = \{1, 2, \dots, n\}$ (X 的元素表示对 n 个城市的编号)的一个排列 $\pi(X = \{v_1, v_2, \dots, v_n\})$,使

$$T_d = \sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_i, v_n) \quad (3.9)$$

取最小值。式中的 $d(v_i, v_{i+1})$ 表示城市 v_i 到城市 v_{i+1} 的距离。

3.4.1 编码与适应度函数

在求解 TSP 问题的各种遗传算法中,多采用以遍历城市的次序排列进行编码的方法。如码串 12345678 表示自城市 1 开始,依次经城市 2, 3, 4, 5, 6, 7, 8, 最后返回城市 1 的遍历路径。显然,这是一种针

对 TSP 问题的最自然的编码方式。这一编码方案的主要缺陷在于引起了交叉操作的困难。

另一种较为常用的编码方案是采用“边”的组合方式进行编码。例如码串24536871的第1个码2表示城市1到城市2的路径在 TSP 圈中,第2个码4表示城市2到城市4的路径在 TSP 圈中,以此类推,第8个码1表示城市8到城市1的路径在 TSP 圈中。这一编码方式有着与前面的“节点”遍历次序编码方式相类似的缺陷。

针对上述缺陷,我们可以采用一种比较复杂的间接“节点”编码方式以消除“一点交叉”策略(或多点交叉策略)引起的非法路径问题^[13]。但由于这种编码方式特征遗传性较差,因此现行的研究中很少采用。

适应度函数常取路径长度 T_d 的倒数,即 $f=1/T_d$ 。若结合 TSP 的约束条件(每个城市经过且只经过一次),则适应度函数可表示为: $f=1/(T_d+\alpha\cdot N_i)$,其中 N_i 是对 TSP 路径不合法的度量(如取 N_i 为未遍历的城市的个数), α 为惩罚系数,常取城市间最长距离的两倍多一点(如 $2.05\cdot d_{\max}$)。

3.4.2 交叉策略

基于 TSP 问题的顺序编码(其它编码如以边的组合状态进行编码也呈现相似特性),若采取简单的一点交叉或多点交叉策略,必然以极大的概率导致未能完全遍历所有城市的非法路径。如8城市的 TSP 问题的两个父路径为

1 2 3 4 | 5 6 7 8

8 7 6 5 | 4 3 2 1

若采取一点交叉,且交叉点随机选为4,则交叉后产生的两个后代为

8 7 6 5 5 6 7 8

1 2 3 4 4 3 2 1

显然,这两个子路径均未能遍历所有8个城市,都违反 TSP 问题

的约束条件。针对这一问题,当然可以采取上述构造惩罚函数的方法,但试验效果不佳。理论分析上可能的解释是,这一方法将本已十分复杂的 TSP 问题更加复杂化了。因为满足 TSP 问题约束条件的可行解空间规模为 $n!$;而按构造惩罚函数的方法,其搜索空间规模变为 n^n ;随着 n 的增大, $n!$ 与 n^n 之间的差距是极其惊人的。解决这一约束问题的另一种处理方法是对交叉、变异等遗传操作作适当的修正,使其自动满足 TSP 的约束条件,目前已有各种各样的这类处理方法,现将常用的几种交叉方法介绍如下:

1. 部分匹配交叉(PMX,partially matched crossover)法

PMX 操作是由 Goldberg 和 Lingle 于 1985 年提出的^[14]。在 PMX 操作中,先依据均匀随机分布产生两个位串交叉点,定义这两点之间的区域为一匹配区域,并使用位置交换操作交换两个父串的匹配区域。考虑下面一个实例,如两父串及匹配区域为

$$A = 9\ 8\ 4\ |5\ 6\ 7\ |1\ 3\ 2\ 0$$

$$B = 8\ 7\ 1\ |2\ 3\ 0\ |9\ 5\ 4\ 6$$

首先交换 A 和 B 的两个匹配区域,得到

$$A' = 9\ 8\ 4\ |2\ 3\ 0\ |1\ 3\ 2\ 0$$

$$B' = 8\ 7\ 1\ |5\ 6\ 7\ |9\ 5\ 4\ 6$$

对于 A' 、 B' 两子串中匹配区域以外出现的遍历重复,依据匹配区域内的位置映射关系,逐一进行交换。对于 A' 有 2 到 5,3 到 6,0 到 7 的位置符号映射,对 A' 的匹配区以外的 2,3,0 分别以 5,6,7 替换,则得

$$A'' = 9\ 8\ 4\ |2\ 3\ 0\ |1\ 6\ 5\ 7$$

同理可得:

$$B'' = 8\ 0\ 1\ |5\ 6\ 7\ |9\ 2\ 4\ 3$$

这样,每个子串的次序部分地由其父串确定。

2. 顺序交叉法(OX,order crossover)法

与 PMX 法相似,Davis(1985)等人提出了一种 OX 法^[15]。此方法开始也是选择一个匹配区域:

$$A = 9\ 8\ 4\ |\ 5\ 6\ 7\ |\ 1\ 3\ 2\ 0$$

$$B = 8\ 7\ 1\ |\ 2\ 3\ 0\ |\ 9\ 5\ 4\ 6$$

并根据匹配区域的映射关系,在其区域外的相应位置标记 H ,得到

$$A' = 9\ 8\ 4\ |\ 5\ 6\ 7\ |\ 1\ H\ H\ H$$

$$B' = 8\ H\ 1\ |\ 2\ 3\ 0\ |\ 9\ H\ 4\ H$$

再移动匹配区至起点位置,且在其后预留相等于匹配区域的空间(H 数目),然后将其余的码按其相对次序排列在预留区后面,得到

$$A'' = 5\ 6\ 7\ H\ H\ H\ 1\ 9\ 8\ 4$$

$$B'' = 2\ 3\ 0\ H\ H\ H\ 9\ 4\ 8\ 1$$

最后将父串 A, B 的匹配区域相互交换,并放置到 A'', B'' 的预留区内,即可得到两个子代:

$$A''' = 5\ 6\ 7\ |\ 2\ 3\ 0\ |\ 1\ 9\ 8\ 4$$

$$B''' = 2\ 3\ 0\ |\ 5\ 6\ 7\ |\ 9\ 4\ 8\ 1$$

虽然,PMX 法与 OX 法非常相似,但它们处理相似特性的手段却不同。PMX 法趋向于所期望的绝对城市位置,而 OX 法却趋向于期望的相对城市位置。

3. 循环交叉(CX, cycle crossover)法

Smith 等人^[16]提出的 CX 方法与 PMX 方法和 OX 方法有不同之处。循环交叉的执行是以父串的特征作为参考,使每个城市在约束条件下进行重组。设两个父串为

$$C = 9\ 8\ 2\ 1\ 7\ 4\ 5\ 0\ 6\ 3$$

$$D = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 0$$

不同于选择交叉位置,我们从左边开始选择一个城市:

$$C' = 9\ -\ -\ -\ -\ -\ -\ -\ -\ -\ -$$

$$D' = 1\ -\ -\ -\ -\ -\ -\ -\ -\ -\ -$$

再从另一父串中的相应位置,寻找下一个城市:

$$C' = 9 \text{ --- } 1 \text{ --- --- --- ---}$$

$$D' = 1 \text{ --- --- --- --- --- } 9 \text{ ---}$$

再轮流选择下去,最后可得

$$C' = 9 \ 2 \ 3 \ 1 \ 5 \ 4 \ 7 \ 8 \ 6 \ 1 \ 0$$

$$D' = 1 \ 8 \ 2 \ 4 \ 7 \ 6 \ 5 \ 1 \ 0 \ 9 \ 3$$

关于 PMX,OX,CX 方法的更进一步的理论和实验分析,可参见 Oliver,Smith,& Holland^[17]。

4. 基于知识的交叉方法

这种方法是一种启发式的交叉方法,可按以下规划构造后代:

(1) 随机地选取一个城市作为子代圈的开始城市。

(2) 比较父串中与开始城市邻接的边,选取最小的边添加到圈的路径中。

(3) 重复第(2)步,如果发现按最小边选取的规划产生非法路径(重复经过同一城市),则按随机法产生一合法的边,如此反复,直至形成一完整的 TSP 圈。使用这一方法,可获得较好的结果。在200个城市的 TSP 优化方面,已产生接近由模拟退火程序(Bonomi & Lutton^[18], Kirkpatrick, Gelatt, & Vecchi^[19]) 计算出的最优结果。不过,这一方法使用了基于问题的一些知识,损失了遗传算法的通用性和鲁棒性。

关于 TSP 问题的遗传交叉方法还有各种各样的变形方法,一般来说,交叉方法应能使父串的特征遗传给子串,子串应能部分或全部地继承父串的结构特征和有效基因。

3.4.3 变异技术

与进化规划不同的是,遗传算法强调交叉的功能(参见第六章)。从遗传算法的观点来看,解的进化主要靠选择机制和交叉策略来完成,变异只是为选择、交叉过程中可能丢失的某些遗传基因进行修复和补充,变异在遗传算法的全局意义上只是一个背景操作。针对 TSP 问题,主要的变异技术如下述。

1. 位点变异

变异仅以一定的概率(通常较小)对串的某些位作值的变异。

2. 逆转变异

在串中,随机选择两点,再将这两点内的子串按反序插入到原位
置中,如串 A 的逆转点为 3,6,则经逆转后,变为 A'

$$A = 1\ 2\ 3\ |4\ 5\ 6\ |7\ 8\ 9\ 0$$

$$A' = 1\ 2\ 3\ |6\ 5\ 4\ |7\ 8\ 9\ 0$$

这种变异操作对于 TSP 问题,就调整前后引起的 TSP 圈的长度变化而言属于最细微的调整,因而局部优化的精度较高;但码串绝对位置所呈现的“模式”变化较大,所需的计算也稍为复杂一点。

3. 对换变异

随机选择串中的两点,交换其值(码)。对于串 A

$$A = 1\ 2\ 3\ 4\ \wedge\ 5\ 6\ 7\ \wedge\ 8\ 9$$

若对换点为 4,7,则经对换后, A' 为

$$A' = 1\ 2\ 3\ 7\ \wedge\ 5\ 6\ 4\ \wedge\ 8\ 9$$

这种变异操作在求解 TSP 问题优化算法中常被采用。在遗传算法中,对换变异操作对码串绝对位置所呈现的“模式”变化影响较小,所需的计算也简单一些,但局部优化精度稍差一点。

4. 插入变异

从串中随机选择 1 个码,将此码插入随机选择的插入点中间,对于上述 A 而言,若取插入码为 5,选取插入点为 2~3 之间,则

$$A' = 1\ 2\ 5\ 3\ 4\ 6\ 7\ 8\ 9$$

此外,还有一些有关上述变异操作的变体形式,如引入连续逆转、进化变异(爬山法)和混合变异等。

3.4.4 选择机制和群体构成

在遗传算法中,最常见的选择机制是依据适应度的比例概率选择机制;在有限规模的群体中,适应度较高的个体有较大的机会繁殖后代,即生物进化论上的适者生存规则。

在新一代群体构成方法方面,有全部替换上一代群体的全刷新代际更新方式(称 N 方式),有保留一个最好的父串的最佳保留(elitist)群体构造方式(称 E 方式),有按一定比例更新群体中的部分个体的部分更新方式(称 G 方式,或称代沟方法,这种情况的极端是每代仅删去一个最不适的个体的最劣死亡方式),也有从产生的子代和父代中挑选最好的若干个个体的群体构成形式(称 B 方式)。从群体规模来看,有变化规模的方式,也有恒定规模的群体构成方式等。

一般讲,N 方式的全局搜索性能最好,但收敛速度最慢;B 方式收敛速度最快,但全局搜索性能最差;E 方式和 G 方式的性能介于 N 方式和 B 方式之间。在求解货郎担问题的应用中,多选用 E 方式。

3.4.5 混合 GA 技术

遗传算法(GA)在局部搜索能力方面的缺陷是为人共知的。Gunter Rudolph^[20],David B. Fogel^[21]等人已就标准遗传算法(基于比例选择机制,二进制编码,一点交叉技术,位点变异策略的遗传算法)的收敛性作了比较详细的分析,得出的结论是,遗传算法是一个渐进的收敛过程。在采用最佳保留选择机制的遗传算法中,全局最优解是有可能搜索到的。但是,在实际应用中,遗传算法一般收敛到某一可行解,但这一可行解也许不是全局最优点,甚至不是局部最优点。因此,在遗传算法框架中,适度地引入其它的局部搜索方法用以改善遗传算法的局部搜索能力在理论和实际上都是十分必要的。Takashi Kido 等人^[22]已将 TABU 技术和 SA 技术引入遗传算法框架中,其实验结果显示,就 TSP 的最终求解质量而言,GA+TABU,GA+SA 以及 GA+TABU+SA 均优于单纯的遗传算法(GA)。有关混合 GA 技术的细节可参阅 3.5 节。

3.4.6 基于遗传算法求解 TSP 的算法实现

我们对遗传算法方法求解 TSP 问题作过实验研究,现就我们所用的遗传算法框架,介绍其算法实现过程。

1. 编码与适应度函数

我们以 n 城市的遍历次序作为遗传算法的编码, 由于在可行解群体的初始化、交叉操作及变异操作中均隐含 TSP 问题的合法性约束条件, 故适应度函数取为哈密尔顿圈的长度的倒数(无惩罚函数)。

2. 选择机制

开始, 我们用随机方法产生初始解群。随着遗传算法的执行, 我们保留 M 个较优的个体作为样本群体, 以供选择; 在每一代运算过程中, 个体被选中的概率与其在群体中的相对适应度成正比。

3. 交叉方法

我们选用的交叉方法与 OX 法有点类似, 现介绍如下:

(1) 随机在串中选择一个交配区域, 如两父串及交配区域选定为

$$A = 1\ 2\ |\ 3\ 4\ 5\ 6\ |\ 7\ 8\ 9$$

$$B = 9\ 8\ |\ 7\ 6\ 5\ 4\ |\ 3\ 2\ 1$$

(2) 将 B 的交配区域加到 A 的前面或后面, A 的交配区域加到 B 的前面或后面得到

$$A' = 7\ 6\ 5\ 4\ |\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9$$

$$B' = 3\ 4\ 5\ 6\ |\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1$$

(3) 在 A' 中自交配区域后依次删除与交配区相同的城市码, 得到最终的两子串为

$$A'' = 7\ 6\ 5\ 4\ 1\ 2\ 3\ 8\ 9$$

$$B'' = 3\ 4\ 5\ 6\ 9\ 8\ 7\ 2\ 1$$

与其它方法相比, 这种方法在两父串相同的情况下仍能产生一定程度的变异效果, 这对维持群体内一定的多样化特性有一定的作用, 实验中也显示了较好的结果。

4. 变异技术

由于在选择机制中采用保留最佳样本方式, 以及引入了“进化逆转”操作技术, 为保持群体内个体的多样化, 我们采取连续多次对换的变异技术, 使可行解有较大的顺序排列上的变化, 以抑制“进化逆

转”的同化作用。变异操作发生的概率取得比较小(1 %左右),一旦变异操作发生,则用随机方法产生交换次数 K ,对所需变异操作的串进行 K 次对换(对换的两码位也是随机产生的)。

5. “进化逆转”操作

引入“进化逆转”操作的主要目的是改善遗传算法的局部搜索能力。所谓局部搜索通常指的是基于邻域的试探搜索方法。在基本遗传算法操作中,交叉操作在可行解空间中动作范围较宽,步伐较大;变异操作由于受“选择”压力的作用,通常也难以发挥局部搜索的功效(特别在遗传算法趋向收敛的后期阶段)。因此,在遗传算法框架中加入适当的、基于邻域的局部搜索机制,构成一种全局搜索和局部搜索相结合的优化搜索算法,对改进优化质量以及提高搜索效率都是很有意义的。

在自然遗传和进化过程中,“逆转”也是一种常见的现象。如一染色体各片断的正常顺序是(1—2—3—4—5—6),在区间2—3和区间5—6发生了两处断裂,断裂片断又以反向顺序插入,于是逆转后的染色体顺序变为(1—2—5—4—3—6)。自然生物遗传上的“逆转”现象有消极的作用(如导致致死因素、不育因素等),也可能产生积极的作用(如导致生物的适应能力增强等)。从进化意义上讲,如果说交叉、变异等遗传操作使子代趋向于拥有较优品质的基因型的话,那么逆转、对换等遗传操作的功能就是使这些基因型及其组合以较优的次序遗传给后代。在针对 TSP 问题的遗传算法中,“逆转”是一种常见的“变异”技术。我们使用的“进化逆转”是一种单方向的(朝着改进的方向)和连续多次的“逆转”操作,即对于给定的串,若“逆转”使串(可行解)的适应度提高,则执行逆转操作,如此反复,直至不存在这样的逆转操作为止。这一操作实际上使给定的串改良到它的局部极点,这种局部爬山能力与基本遗传算法的全局搜索能力相结合在实验中显示了较好的效果。

6. 算法的流程框图

一个包含选择、交叉、变异及进化逆转的遗传算法流程框图如图

3.26所示。

7. 实验结果

我们按照上述算法编制了相应软件,并在386微机上进行了测试。实验中,群体规模定为100,交叉概率为0.95,变异概率为0.003,初始可行解群体由随机法产生。实验结果表明:

(1) 当 $n \leq 15$ 时,随机样本实验表明,本算法可100%搜索到用穷举法求得的最优解。

(2) 当 $15 \leq n \leq 30$ 时,我们对一组样本进行了测试,结果表明本算法能收敛到一稳定的“最好解”(难以确认其最优性);多次实验的误差结果为0;模拟退火法也可找到相同的“最好解”,但运行时间约为遗传算法的6倍

(3) 对 $n=50$, $n=60$, $n=80$ 及 $n=100$ 的测试结果表明,遗传算法在求解质量上略优于模拟退火法($\alpha=0.95$),优化效率高于模拟退火法。表3.4所示为遗传算法(GA)与模拟退火方法(SA法)的比较实验结果,表中所列 TSP 路径长度为相对长度,其值由下式算出:

$$T_d = T_a / (0.765 \cdot X \cdot \sqrt{N})$$

上式中, T_d 为实际路径的长度, X 为包含 TSP 所有城市的最小正方形的边长, N 为 TSP 问题的城市数目。图3.27为城市规模 $N=100$ 的 TSP 问题的初始群体中的最佳路径,图3.28为经本算法优化得到的最佳路径。

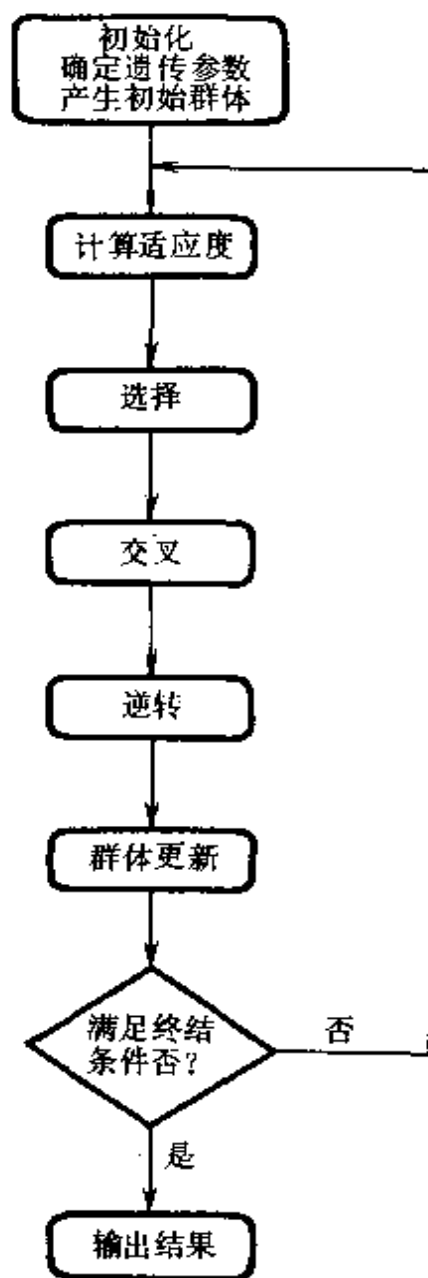


图 3.26 求解 TSP 问题的遗传算法框图

表 3.4 SA 法和 GA 求解 TSP 问题的实验结果

| 城市数 | 最佳解 | 最佳解 | 时间 | 时间 |
|-----|--------|--------|------|-----|
| n | SA 法 | GA 法 | SA | GA |
| | (相对值) | (相对值) | (s) | (s) |
| 50 | 106.33 | 105.88 | 540 | 98 |
| 60 | 105.11 | 104.22 | 480 | 120 |
| 80 | 103.22 | 101.34 | 600 | 150 |
| 100 | 99.11 | 98.05 | 1080 | 360 |
| 200 | 98.67 | 97.91 | 3400 | 660 |

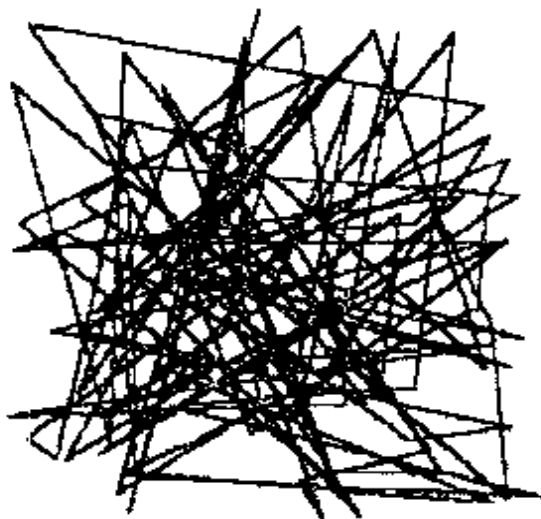


图 3.27 100城市 TSP 问题的
初始最佳路径($G=0$ 代)

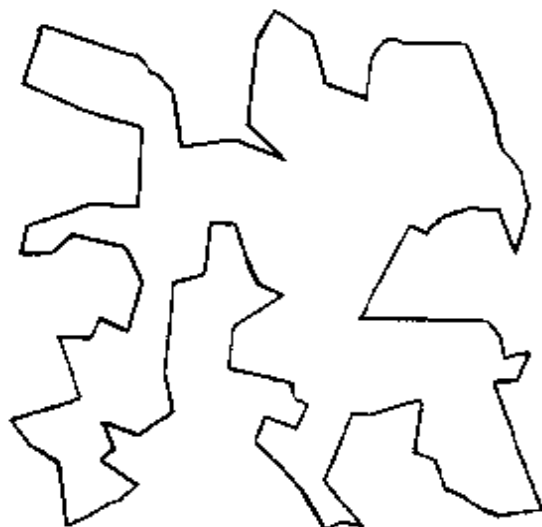


图 3.28 100城市 TSP 问题经遗传算法
优化后得到的最佳路径($G=200$ 代)

3.5 混合搜索方法

3.5.1 概述

基本遗传算法存在着局部搜索能力较差的缺陷。解决这个问题

的一种方法是对基本遗传算法进行适当的改进,如在基本遗传算法中增加交叉约束算子,使交叉操作限制在基因型(编码串)相似的染色体之间就能一定程度地改善遗传算法的局部搜索能力;另一种方法是将遗传算法与传统的、基于问题知识的启发式搜索技术相结合构成基于遗传算法的混合搜索算法框架。如针对“图的划分”问题将遗传算法(GA)与传统的最小分割法相结合(丸山,1992),在神经网络学习算法中将GA与BP算法相结合构成GA-BP混合算法(北野,1991^[22])等均在一定程度上改进了单纯遗传算法的性能(如求解质量、收敛速度等)。本节着重介绍遗传算法与通用性较强的启发式搜索算法爬山法(HC法)、模拟退火法(SA法)以及列表寻优法(TABU法)相结合构成的混合遗传算法,并针对TSP问题介绍其具体实现方法并给出实验结果。

3.5.2 启发式搜索法简介

1. 爬山法(HC法)

爬山法是一种基于邻域搜索技术的、延着有可能改进解的质量的方向进行单方向搜索(爬山)的搜索方法。其基本实现步骤如下(以TSP问题为例):

- (1) 首先随机选择一条TSP合法路径作为当前路径。
- (2) 对当前路径进行一次邻域搜索操作(如对换、逆转操作等)。
- (3) 判断所施操作是否使当前路径长度变短,若是,则以所施操作后改变了的路径作为新的当前路径。
- (4) 重复(2)和(3),直到所有的邻域操作均不能改进解的质量为止。

显然,对于像TSP这类的多峰问题,爬山法由于缺乏问题可行解空间的全局性采样,因此陷入局部解的可能性极大。

2. 模拟退火法(SA法)

模拟退火法是一种基于热力学的退火原理建立的随机搜索算法。为了克服爬山法极易陷入局部解的缺点,模拟退火法使用基于概

率的双方向随机搜索技术;当基于邻域的一次操作使当前解的质量提高时,SA 接收这个被改进的解作为新的当前解;在相反的情况下,SA 以一定的概率 $\exp(-\Delta c/T)$ 接收相对当前解来说质量较差的解作为新的当前解。其中, Δc 为邻域操作前后解的质量差, T 为退火过程的控制参数。模拟退火法已在理论上被证明是一种以概率1收敛于全局最优解的全局优化算法,但其参数难以控制,其主要问题有以下三点:

(1) 温度 T 的初始值设置问题。

温度 T 的初始值设置是影响 SA 全局搜索性能的重要因素之一。初始温度高,则搜索到全局最优解的可能性大,但因此要花费大量的计算时间;反之,则可节约计算时间,但全局搜索性能可能受到影响。实际应用过程中,初始温度一般需要依据实验结果进行若干次调整。

(2) 退火速度问题。

SA 的全局搜索性能也与退火速度密切相关。一般来说,同一温度下的“充分”搜索(退火)是相当必要的,但这需要计算时间。实际应用中,要针对具体问题的性质、特征设置合理的退火平衡条件。

(3) 温度管理问题。

温度管理问题也是 SA 难以处理的问题之一。当邻域搜索过程中,解的质量变差的概率呈 Boltzmann 分布时,S. Geman 和 D. Geman 从理论上证明对数降温方式可使 SA 收敛于全局最优解,即

$$T(t) = k/\log(1+t) \quad (3.10)$$

式中 k 为正的常数, t 为降温的次数。

当邻域搜索过程中,解的质量变差的概率呈 Cauchy 分布时,H. Szu 和 R. Hartley 从理论上证明按式(3.11)的降温方式可使 SA 收敛于全局最优解,即

$$T(t) = k/(1+t) \quad (3.11)$$

式中 k 为正的常数, t 为降温的次数。

实际应用中,由于必须考虑计算复杂度的切实可行性等问题,常

采用如下所示的降温方式:

$$T(t+1) = k * T(t) \quad (3.12)$$

式中 k 为正的略小于1.00的常数, t 为降温的次数。

· 现以 TSP 问题为例,介绍求解 TSP 问题的 SA 方法。算法3.3 为用 SA 法求解 TSP 问题的高层描述。

算法3.3 求解 TSP 问题的 SA 算法

```
begin
initialize(); {初始温度  $T_0$ , 产生初始路径  $P$ }
while(终止条件未满足)
    while(未达平衡态)
        Generate-next-move(); {当前路径的邻域操作}
        If(Accept(Temperature, change-in-cost))
            then Update-tour(); {修改当前路径}
        endwhile;
        Calcilate-new-temperature(); {降温}
    endwhile;
output-tour(); {输出解}
end.
```

算法3.4 Accept 算法

```
procedure accept( $T, \Delta C$ );
begin
    if( $\Delta C < 0$ ) then return(TRUE);
        else if( $\exp(-\Delta C/T) > \text{random}(0,1)$ ) then return(TRUE);
            else return(FALSE);
        endif;
    endif;
end;
```

算法3.3 中 Generate-next-move()操作可用城市位置对换法或

逆转法产生,降温可用式(3.12)处理,通过适当的调整 SA 控制参数,可获得质量相当高的最终解。

3. 列表寻优法(TABU 法)

列表寻优法是解决组合优化问题的另一种优化算法。在这种技术中,首先按照随机方法产生一初始可行解作为当前解,然后搜索当前解的邻域中的所有可行解,取其最好的可行解作为新的当前解。为了避免陷入局部解,这种优化方法允许一定的上山操作(解的质量变差)。另外,为了避免搜索路径的往返重复, TABU 方法使用列表的形式记录搜索路径的历史信息,这可在一定程度上使搜索过程避开局部极值点,开辟新的搜索区域。这一优化方法的主要问题是列表大小(TABU list size)不易确定。一般来说,太小的列表可能无法避免搜索路径的往返重复,这将影响 TABU 方法的全局优化性能;另一方面,过大的列表除了增加计算时空复杂度外,还可能会因列表对搜索区域的过分限制,而使 TABU 搜索难以接近全局最优解的近旁,这又从另一方面影响了 TABU 法的全局搜索能力。实用过程中,列表大小的经验调整是非常必要的。图3.29为 TABU 法的算法框图。

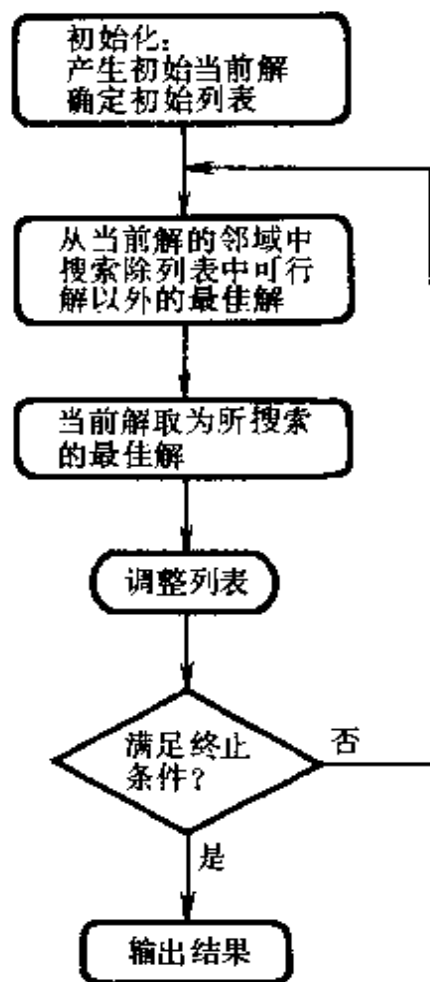


图 3.29 TABU 的算法框图

3.5.3 混合遗传算法(Hybrid GA)

将遗传算法与其它启发式的搜索算法相结合构成混合遗传算法的主要目的是改善基本遗传算法的局部搜索能力,进一步提高优化质量和搜索效率,以弥补单一优化方法的某些不足之处。本小节介绍

遗传算法(GA)与模拟退火法(SA)和列表寻优法(TABU)相结合的方法,并依据实验结果对混合遗传算法的搜索性能进行一些比较和讨论。

GA 与 SA、TABU 等启发式搜索方法的结合方式多种多样,这里介绍的 GA+SA+TABU 混合算法是一种交替式的混合方法,其优化步骤如下:

(1) 初始化,用随机方式产生若干个彼此不同的可行解组成初始可行解群体。

(2) 针对可行解群体中的一半个体,执行 TABU 搜索,求得局优解。

(3) 针对可行解群体中的另一半个体,执行 SA 搜索,求得局优解。

(4) 针对(2),(3)两步求得的局优解,执行选择、交叉等遗传操作。

(5) 重复(2),(3),(4)搜索过程,直到算法的终止条件满足为止。

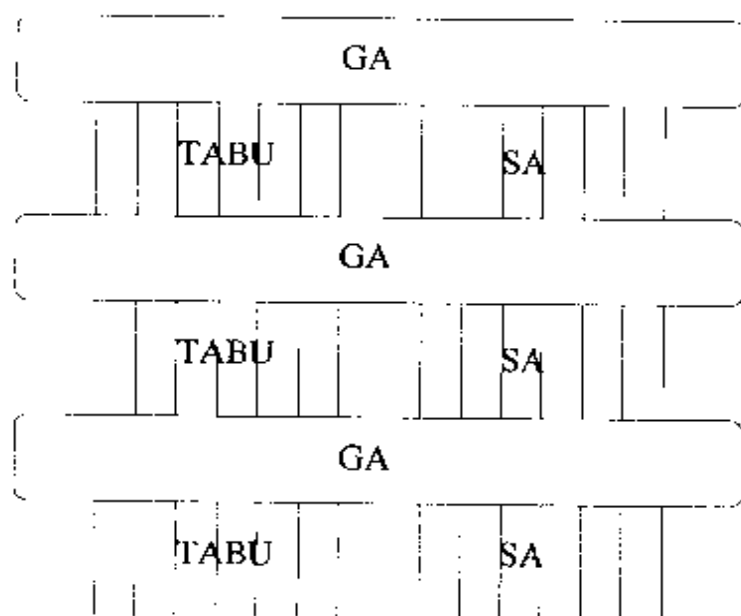


图 3.30 GA+SA+TABU 混合算法原理框图

图3.30为 GA+SA+TABU 优化算法的混合原理框图。

3.5.4 实验与讨论

各种搜索方法的实验结果如表3.5 所示。

表 3.5 各种搜索方法的实验结果

| 优化算法 | 最好解(英里) | 平均值(英里) | 标准偏差 | 计算时间(s) |
|---------------------|---------|---------|------|-----------|
| GA | 22253 | 23316 | 514 | 640(290代) |
| TABU | 21352 | 21433 | 64.8 | 210 |
| SA($\alpha=0.63$) | 21331 | 21372 | 30.3 | 310 |
| SA($\alpha=0.90$) | 21255 | 21284 | 27.5 | 1340 |
| GA+SA+TABU | 21247 | 21247 | 0 | 420(5代) |

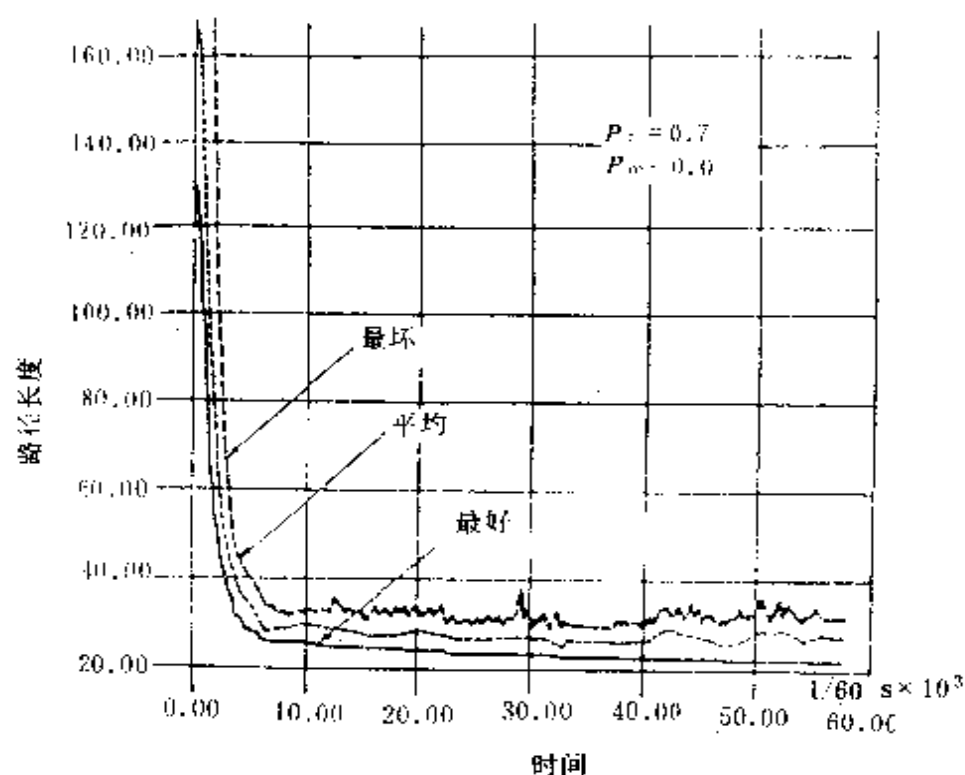


图 3.31 GA 优化过程中,TSP 路径长度随时间变化曲线

1. GA 的搜索能力

图3.31显示了 TSP 的路径长度最小值、平均值及最大值随遗传代数变化的曲线,实验中,群体的规模为100。图3.32为不同群体规模对 GA 性能的影响之变化曲线。在 GA 搜索的初期阶段,收敛的速度比较快,但随着时间的进展,GA 的搜索效率变低。规模大的情形收敛速度较慢,但解的质量要高一点。与 SA、TABU 相比,GA 的效率及求解质量要差一些。

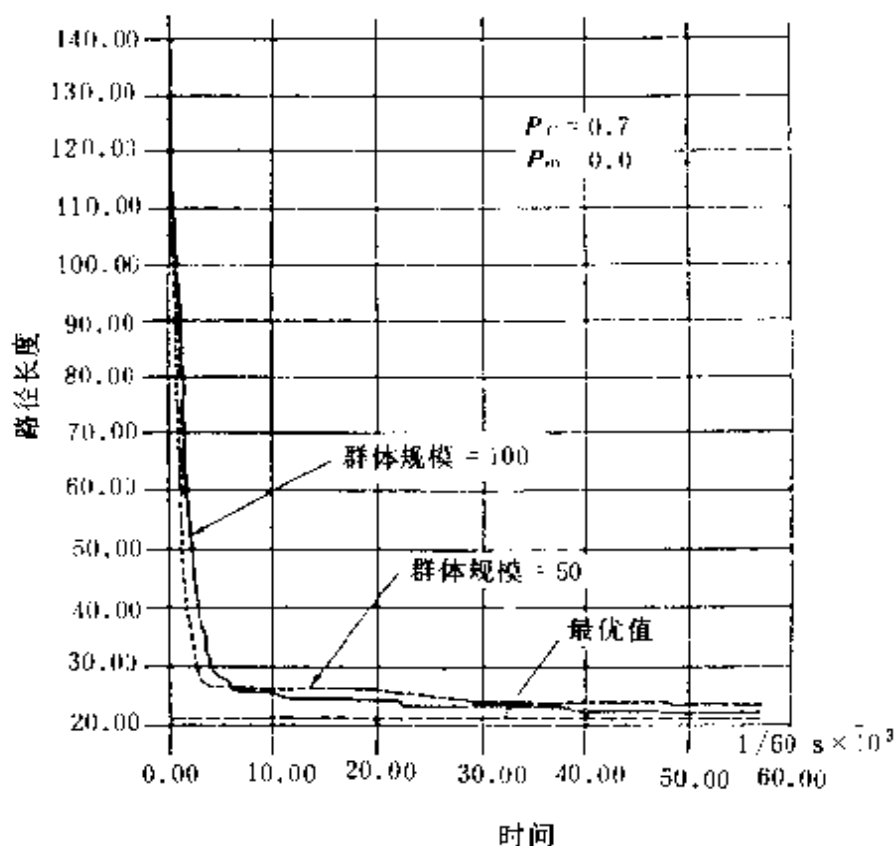


图 3.32 群体规模对 GA 优化性能的影响

2. SA 的搜索能力

图3.33表示了初始温度 $T=1500$,降温系数为0.90时,TSP 的路径长度随时间的变化曲线。一旦 SA 终结,程序自新的初始路径继续 SA 搜索。由图3.33可见,SA 的解的质量比较高,但温度管理比较麻烦,常需实验的多次调整。图3.34为降温系数分别为0.63和0.90时 SA 性能的比较曲线。

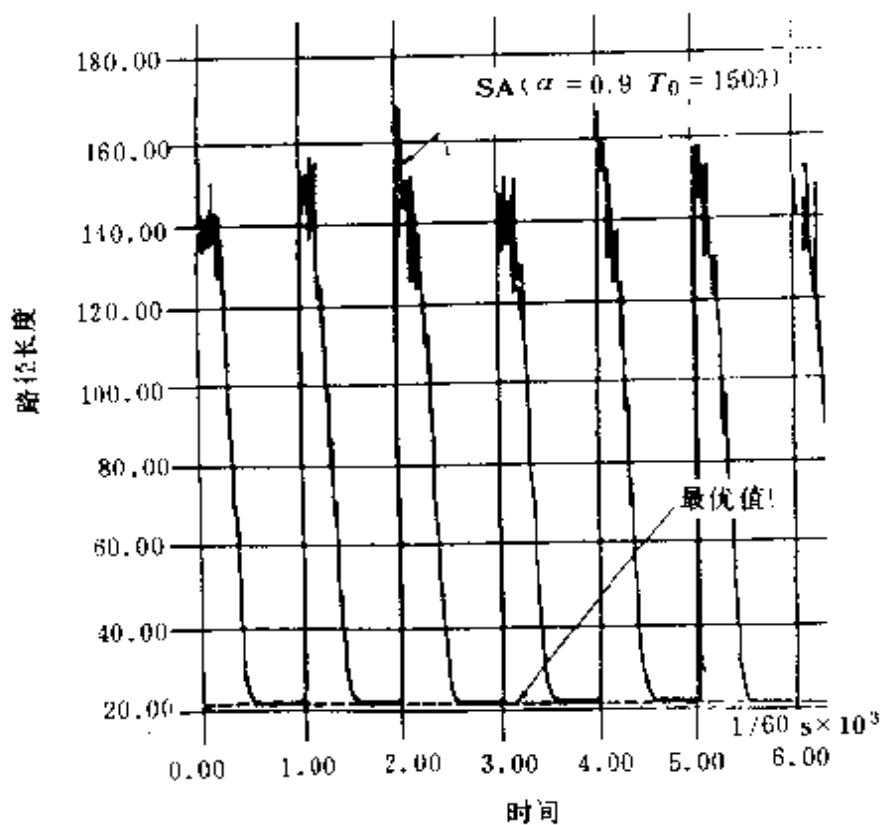


图 3.33 SA 优化过程中, TSP 路径长度随时间变化曲线

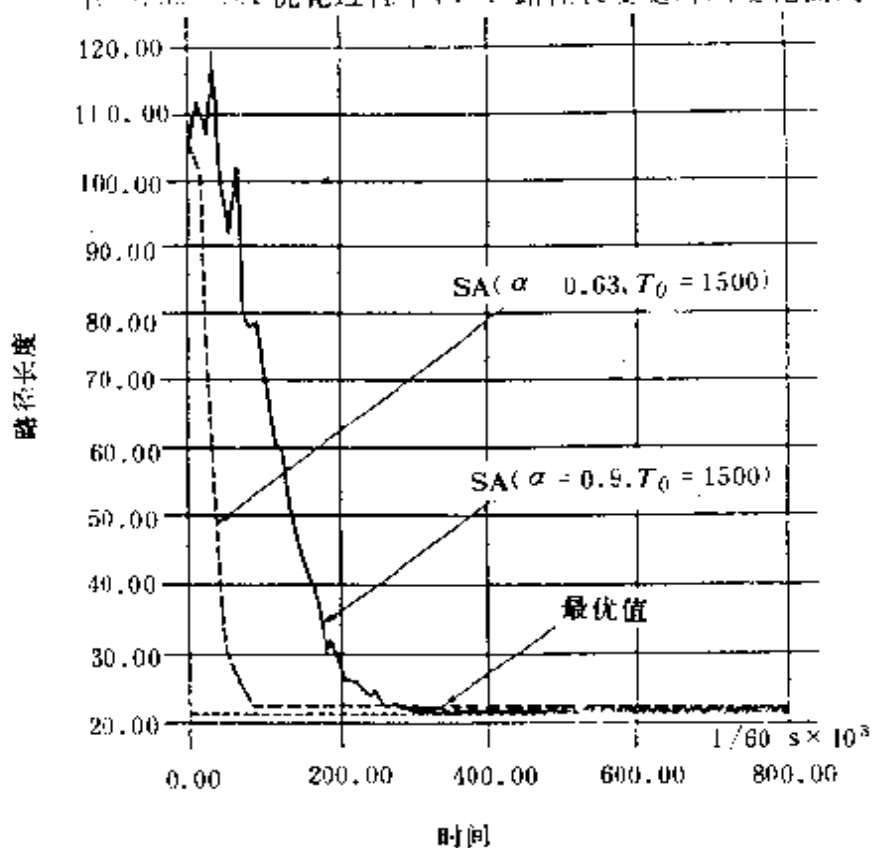


图 3.34 降温系数对 SA 优化性能的影响

3. TABU 的搜索能力

图 3.35 为 TABU 方法的 TSP 的路径长度变化曲线,其中 TABU 的列表规模为 5。图 3.36 为 SA 与 TABU 的比较实验曲线。TABU 比 SA 有较早收敛的倾向,但 SA 在合理的温度管理情况下,解的质量较高。

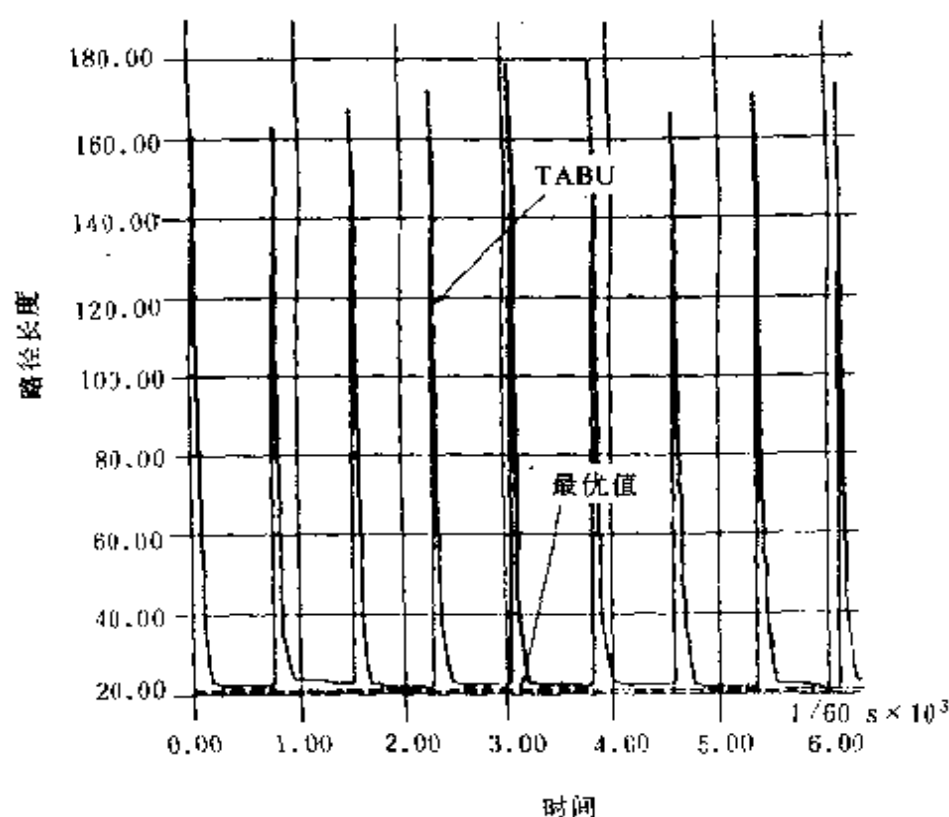


图 3.35 TABU 优化过程中, TSP 路径长度随时间变化曲线

4. GA+SA+TABU 的搜索能力

GA+SA+TABU 与单纯 GA 的性能比较如图 3.37 所示。GA 大范围的搜索能力较强,但局部搜索能力较弱;针对单纯 GA 的弱点,GA+SA+TABU 方法强化了单纯 GA 的局部搜索能力。与单纯 SA 方法、单纯 TABU 方法相比,GA+SA+TABU 方法拓展了 SA 及 TABU 的局部搜索范围,增强了单纯 SA, TABU 的全局搜索能力。实验结果表明,GA+SA+TABU 混合算法与单纯 GA, SA, TABU 相比,收敛速度要快得多,解的质量也最高。

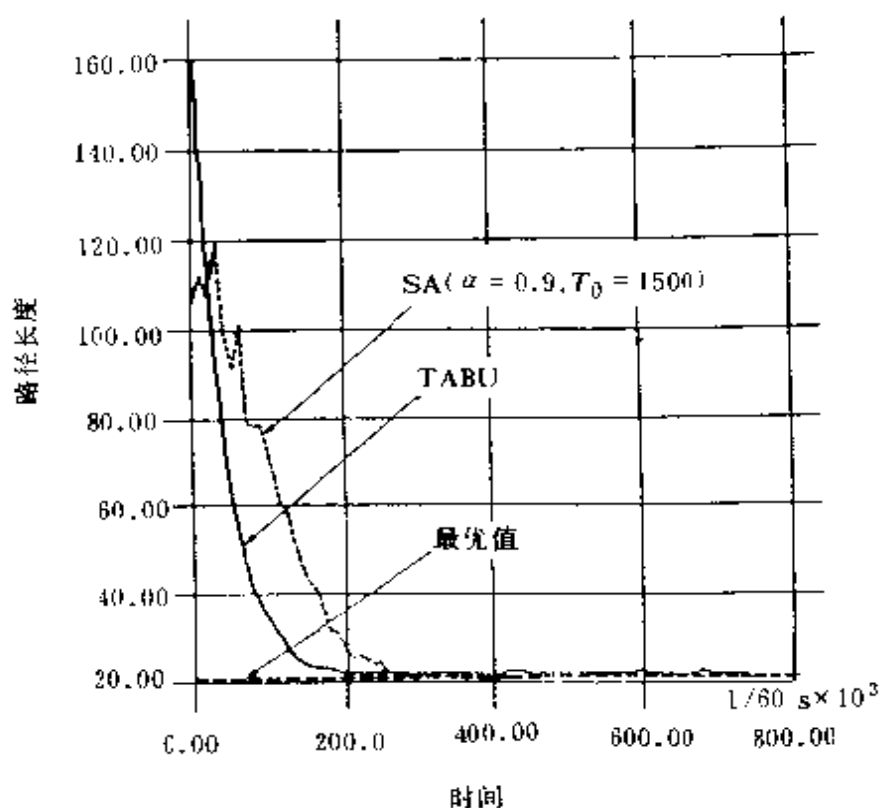


图 3.36 SA 法与 TABU 法的搜索性能比较

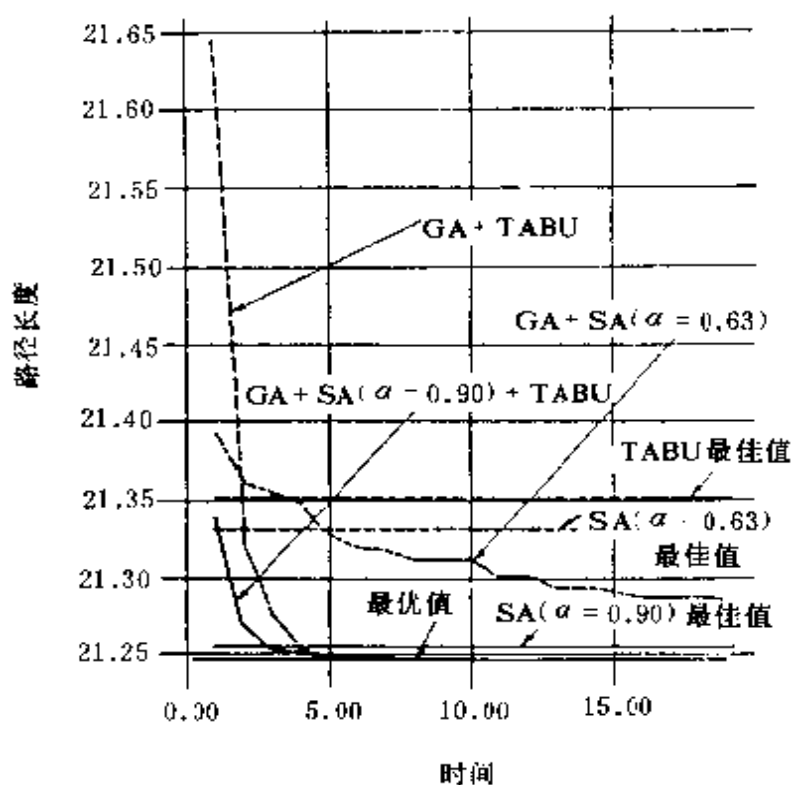


图 3.37 各种优化方法的搜索性能比较

3.6 图的划分问题

图的划分问题在集成电路及印刷电路板的自动布局等 CAD 应用领域有着极高的应用价值。本节将介绍基于遗传算法处理图的划分问题的优化方法。

3.6.1 问题描述

图的二划分问题：对于一无向图 G ，设其顶点集合为 V ，将顶点集合 V 划分为两个子集 V_1 和 V_2 ， $V_1 \cap V_2 = \emptyset$ ，求使 V_1 和 V_2 两顶点子集之间联结最少的一种划分。

图的划分问题在电子线路设计中非常重要。例如，在多层印刷电路板的布局设计中，使层间联线数目最少的器件布局等。由于图的划分问题的计算复杂度极高（3000个节点的二划分问题的搜索空间可达 10^{900} ），因此，在实用规模上精确求出最优解是不可能的。

针对图的二划分问题，已提出多种启发式的搜索方法；其中，最小分割（Min—Cut）法及其各种各样的改良版本^[23~25]在实际应用中用得较多。其基本思想是寻找当前二划分子集中与其它各节点连接最少的节点，将其移动到另外的子集中，如此反复移动，直到可行解不能改进为止。这种方法虽求解质量相对较高，但对初始划分的依赖性很强，陷入局部解的可能性较大。另一方面，遗传算法虽全局搜索能力较强，但也存在局部搜索能力较弱的缺陷。解决这一问题的手段之一是将遗传算法与最小分割法相结合构成混合遗传算法优化图的二划分问题。

3.6.2 编码与适应度函数设计

图的二划分问题可采用二进制编码。例如个体 01011101 表示将顶点集合 1, 2, 3, 4, 5, 6, 7, 8 划分为 1, 3, 7 和 2, 4, 5, 6, 8 两个子集。

目标函数值为所划分的两子集之间的联结数 k 。适应度可设计

为:

$$f = c - k$$

其中 c 为某一合适的整数常量。

3.6.3 遗传操作

1. 选择机制

可选比例选择机制,或排序选择机制。

2. 交叉策略

可选一点交叉或多点交叉策略。

3. 变异

可选常规的位点变异方案。对于图的二划分问题,由于码串0101与码串1010的划分相同,因此,采用位间交换变异技术也是一种常用的方案。

3.6.4 实验结果

图3.38比较了4种优化方法的最佳解曲线。其优化对象均为3096个节点的图的二划分问题,所用机器为 Symmetry,可用的处理器数为15,实验结果均为50次实验的平均值,所用的4种优化方法分别是:

1. APGA 法(异步并行最小分割——GA 混合算法)(15PE)
2. SPGA 法(同步并行最小分割——GA 混合算法)(15PE)
3. MC 法(并列最小分割法)(15PE)
4. HCME(并列阶梯型最小分割法)(15PE)

APGA 法和 SPGA 法的主要差异在于:SPGA 法要求只能在所有的处理器对各个体所实施的最小分割法均处于终结状态时才能执行遗传操作;而 APGA 法则无此限制。HCME 法^[26]是一种 MC 法的改良算法。实验结果显示,APGA 法的优化效率最高,达到最佳解的时间约为 SPGA 法的一半;在求解质量方面,APGA 法和 SPGA 法均优于 MC 法及 HCME 法。

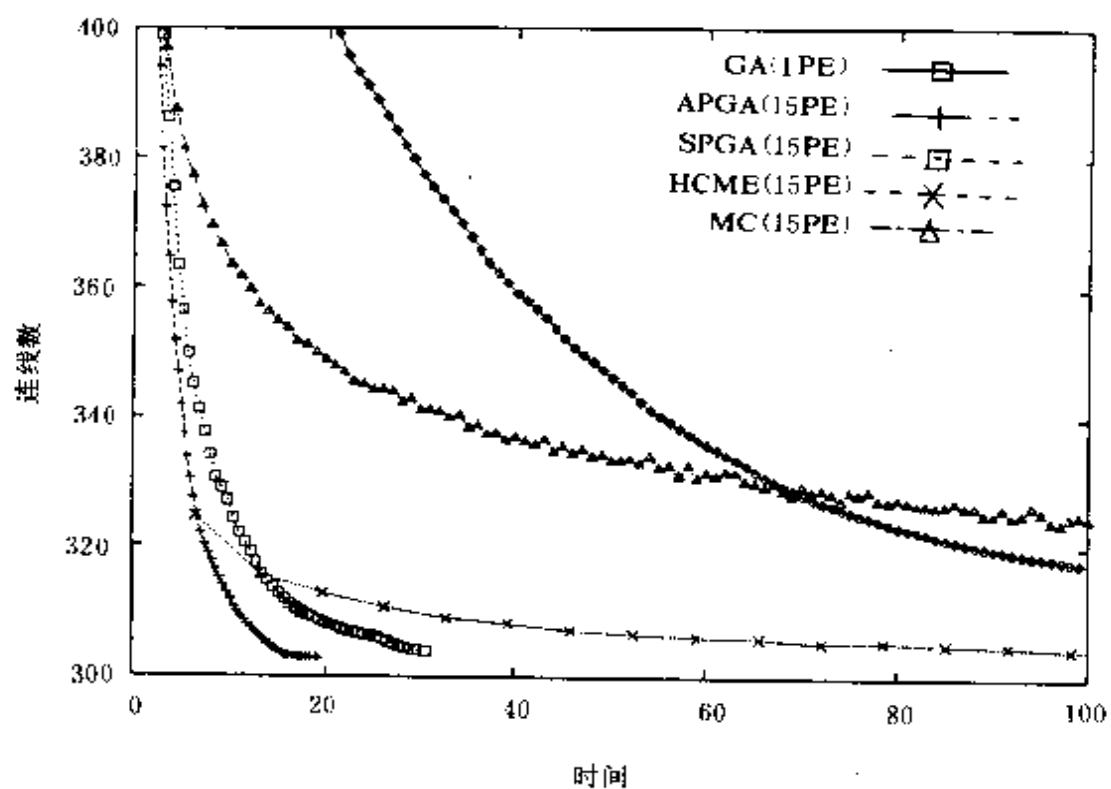


图 3.38 针对图的划分问题的各种算法的搜索性能比较

参 考 文 献

- [1] Bagley J D. The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms. Doctoral Dissertation, University of Michigan, 1967. 136
- [2] Rosenberg R S. Simulation of Genetic Populations with Biochemical Properties. Doctoral Dissertation. University of Michigan, 1967
- [3] Hollstien R B. Artificial Gentic Adaptation in Computer Control Systems. Doctoral Dissertation. University of Michigan, 1971
- [4] Brindle A. Genetic Algorithms for Function Optimization. Doctoral Dissertation. Universion of Alberta, 1981
- [5] Goldberg D E & Smith R E. Nonstationary Function Using Genetic Algorithms with Dominance and Diploidy. Genetic Algorithms and Their Applitations; Proceedings of the Second International Conference on Genetic Algorithms, 1987. 59~68
- [6] Goldberg D E & Smith R E. AI meets OR; Blind Inferential Search with Genetic Algorithms. Paper Presented at the OR-SA/TIMS Joint National Meeting. Miami, FL, 1986
- [7] Cavicchio D J. Adaptive Search Using Simulated Evolution; Doctoral Dissertation. University of Michigan, Ann, Arbor, 1970
- [8] Cavicchio D J. Reproductive Adaptive Plans. Proceedings of the ACM 1972 Annual Conference, 1972. 1~11
- [9] Glodberg D E. Richardson J. Gentic Algorithms with Sharing for Multimodal Function Optimization. Genetic Algorithms and Their Applications; Proceedings of the Second Internation-

- al Conference on Genetic Algorithms, 1987. 41~49
- [10] Grefenstette J J. Optimization of Control Parameters for Genetic Algorithms IEEE Trans. on Systems, Man, & Cybernetics, 1986, 16(1)
- [11] Booker L B. Intelligent Behavior as an Adaptation to the Task Environment; Doctoral Dissertation. University of Michigan, Ann, Arbor, 1982
- [12] Fogel D B. Applying Evolutionary Programming to Selected Traveling Salesman Problems. Cybernetics and Systems, 1993, 24: 27~36
- [13] Grefenstette J J, et al. Genetic Algorithms for the Traveling Salesman Problem. Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985. 160~168
- [14] Goldberg D E, & Lingle R. Alleles, loci, and the Traveling Salesman Problem. Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985. 154~159
- [15] Davis L. Job Shop Scheduling with Genetic Algorithms. Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985. 136~140
- [16] Smith D. Bin Packing with Adaptive Search. Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985. 202~206
- [17] M Oliver L, Smith D J, Holland J R C. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. Genetic Algorithms and Their Applications; Proceedings of the Second International Conference on Genetic Algorithms, 1987. 224~230

- [18] Bonomi E, Lutton J L. The N-city Traveling Salesman Problem; Statistical Mechanics and Metropolis Algorithm. SIAM Review, 1984, 26(4): 551~569
- [19] Kirkpatrick S, Gelatt C D, Vecchi M P. Optimization by Simulated Annealing. Science, 1983, 220(4598): 671~680
- [20] Rudolph G. Convergence Analysis of Canonical Genetic Algorithms. IEEE Trans. on Neural Networks, 1994, 5(1): 96~101
- [21] Fogel D B. Asymptotic Convergence Properties of Genetic Algorithms and Evolutionary Programming; Analysis and Experiments, Cybernetics and Systems, 1994(25): 389~407
- [22] 北野宏明. 遺伝的アルゴリズム. 産鄴図書, 1993
- [23] Laszewski V, Gregor. Intelligent Structural Operators for the K-way Graph Partitioning Problem. ICGA' 91, Morgan Kaufman, 1991. 45~52
- [24] Kernighan B W, Lin S. An Efficient Heuristic Procedure for Partitioning Graphs. Bell Systems Technical Journal, 1970, 49: 291~307
- [25] Cohoon J P, Martin W N, Richards D S. A Multi-population Genetic Algorithm for Solving the K-partition Problem on Hyper-cubes. ICGA' 91, Morgan Kaufman, 1991. 244~248
- [26] Edahiro M, Yoshiyama T. New Placement and Global Routing Algorithms for Standard Cell Layouts. Proceedings of 27th DAC, 1990. 42~645

第四章 遗传算法与机器学习

为解决专家系统设计中的知识获取瓶颈问题而兴起的机器学习研究,其目标之一是能够实现知识的自动获取。时至今日,机器学习的研究仍方兴未艾,其方法也是多种多样。遗传算法作为模拟生物界中的自然选择与生物遗传机制的一种搜索算法,从其开始就与机器学习有着密切联系。分类器系统 CS-1 是遗传算法的创立者 Holland 教授等实现的第一个基于遗传算法的机器学习系统。本章 4.2 节介绍的是一种通用性的分类器系统的体系结构,它由规则与消息子系统、信用分配系统及遗传算法三部分构成,其中信用分配系统是基于 Holland 在 1986 年提出的桶队算法。4.3 节介绍了一个由 Smith 1980 年实现的学习系统 LS-1,它在染色体表示、搜索结构的形成等方面与一般的分类器系统有着重要的差别,完全回避了信用分配问题。4.4 节结合 De Jong 的 GABIL 系统介绍了基于遗传算法的概念学习的基本方法,包括概念空间的染色体表示方法,标准及与领域相关的遗传操作实现等。

4.1 概 述

根据学习策略的不同,机器学习可划分为归纳学习、类比学习、基于解释学习和发现式学习等。而基于机器学习方法的自动知识获取研究正在向纵深发展。虽然从根本上说机器学习方法的应用仍较为困难,但在某些方面已逐步走向成熟,最具代表性的是从例子中进行概念学习,其任务是从概念的特殊例子中归纳出概念的一般性描述,较著名的方法有 ID3^[1]、决策规则归纳 AQ 系列^[2]等。

实际上,概念学习可以看作是对概念描述空间的一种启发式搜

索。概念描述空间是对原始数据(即由教师或环境向学习系统提供的某些概念的实例)使用一定推理规则得到的。概念学习中所隐含的这种搜索机制以及它所采用的符号表示方法,使得遗传算法在概念学习领域有其用武之地。遗传算法本身固有的鲁棒性,使得基于遗传算法的概念学习系统具有更少的限制性。因此近年来在这一方面引起人们的很大兴趣,并开展了相应的研究。

1978年 Holland 等实现了第一个基于遗传算法的机器学习系统第一级认知系统 CS-1(Cognitive System Level One)^[3],它被用来解决两个迷宫问题(maze-running)。该系统由消息表(message list)与称为分类器(classifier)的简单字符串规则、遗传算法及一个报偿分配机制组成。后经 Holland 不断探索,在1986年又提出一种称为桶队(bucket brigade)算法^[4]的反馈机制,该系统被称为分类器系统。因其体系结构某种程度上的通用性,使得它在基于 GA 的机器学习中影响较大。另一种值得一提的是 LS-1^[5]系统,它是由 Smith 在1980年实现的一种基于 GA 的学习系统。虽然它被人们认为是 CS-1的后继,但在某些重要方面,如染色体的表示、反馈方式等,LS-1和 CS-1却差异明显。CS-1和 LS-1的两种不同染色体表示分别被人们称为 Michigan 方法和 Pittsburgh 方法。

CS-1与 LS-1的成功对分类器系统的深入研究起了推动作用,1982年,Booker 在他的博士论文中对分类器系统,自然智能与人工智能的相互关系作了研究^[6]。其分类器系统直接源于 CS-1,但对其体系结构作了改变。Wilson 研究了一种用于协调可移动式视频摄像机的感知-运动(sensory-motor)的分类器系统(称 EYE-EYE)^[7],其主要任务是学习如何通过移动摄像机,将某个目标置于视频摄像机的视觉场(vision field)中心。从体系结构上看,Wilson 的 EYE-EYE 与 CS-1相似,但在规则结构上 Wilson 没有采取一维的字符串模式来表示分类器条件,而是采用了 4×4 的矩阵形式,并设计出了一种相应的棋盘交叉(checkboard crossover)操作。此后,受 Booker 工作的启发,他又研制了 ANIMAT^[8]系统,它是一个在工作林地

(wood)中寻找食物和避开树木的游历(wandering)分类器系统,整个二维空间布局成一 18×58 的长方形网格,每一小块林地由树簇(tree cluster)及食物组成,具有规整的形状。处在某一块林地中的ANIMAT知道与自己紧邻的四周的情况。Wilson在具体实现中引入了一种隐桶队(implicit bucketbrigade)的信用分配机制以及对分类器子群体大小进行控制的共享(sharing)机制。ANIMAT通过遗传算法来学习如何根据周围情况来移动自己,以尽可能少的时间步(time steps)来获得食物。Goldberg研究了用分类器系统来学习控制一个煤气管道仿真系统^[9],他的工作主要由两部分组成,一是通过遗传算法来优化管道的操作,二是通过分类器系统学习控制管道操作。

从应用角度来说,这些系统对顺序决策这类学习问题较为合适。该类问题可以描述如下:一决策主体以回复方式与一具有离散时间状态的动态系统交互,在每个时间步的开始,系统处于某确定状态,该主体依当前状态,根据决策规则,从有限的动作集中选择一个动作供动态系统执行,并进入到一个新的状态,同时向主体反馈一个补偿(payoff),其目的是发现一决策规则集以使补偿最大化。

本章将在4.2和4.3节分别介绍分类器系统CS-1与学习系统LS-1。在这之后我们将结合GABIL系统^[10]介绍基于遗传算法的概念学习方法。它采用的是染色体表示的Pittsburgh方法及标准的遗传操作。目前有的基于遗传算法的概念学习研究试图将概念学习特有的操作符遗传算法化,如Janikow的GIL^[11]。此外,Green等的COGIN系统采用了染色体表示的Michigan方法^[12],其运行机制与标准GA相差甚远,基本原则是将训练集覆盖既用作一种显式约束以限制概念复杂性,又用来驱动学习系统获得概念的多样性。

4.2 分类器系统

分类器系统是一种学习字符串规则(又称分类器)的学习系

统^[3,13],它由规则与消息(rule and message)系统、信用分配(appor-tionment of credit)系统及遗传算法三个主要部分组成,其中规则与消息系统是产生式系统的一种特殊形式.我们知道,产生式规则的一般形式为:IF <condition> THEN <action>.表面上看,这种知识表示方法有很大的局限性,实际上产生式系统具有计算完备性,且其描述也较方便,一条规则或一个规则集往往能将一种复杂的情况非常紧凑地描述出来.因而它为众多专家系统所采用.但对于学习系统来说,产生式规则的语法过于复杂,不易使用,所以在分类器系统中,对产生式规则的语法作了很大的限制,采用了定长的表示形式,从而适于采用遗传操作。

与传统的专家系统在每一匹配周期采用的单条规则激活这种串行方式不同,分类器系统采用了并行激活方式,即在每一匹配周期,它允许多条规则被同时激活,只有在出现两个互斥的动作或当匹配的规则集大小超出消息表的容量时,才考虑规则的选择问题。

在传统的专家系统中,规则的相对强度是由设计者会同领域专家来人为确定的,而在分类器系统中,它却是一种关键的学习对象。为实现这种学习,分类器系统中的分类器都被置于一种“基于信息的服务经济体系”(information-based service economy)中。在对相关消息作出响应的候选分类器之间展开竞争,实际响应者为叫价(bid)最高者,与发送该消息有关的分类器的强度要随之增加,其增加量为响应消息的分类器的叫价。这样,分类器就通过这种响应与激活方式形成了一条连接“制造商”(即探测器)到“消费者”(环境动作和报偿)的分类器链(又称“中间商链”),其间蕴含的竞争机制确保了好的规则最终保留下来,不好的规则逐渐被淘汰。这种通过环境向分类器提供报偿形式的反馈所实现的学习机制通常称为信用分配。系统将根据规则的强度来评价哪些规则是有效的。本节后面将介绍一种典型的信用分配方法——桶队算法(bucket brigade algorithm)。

分类器系统在对规则作出评价之后,还要能利用已有经验产生新规则以取代现有不好的规则,该系统采用了遗传算法来实现这种

功能。根据每条规则的适应度来选择、重组和获得新规则并取代某些旧规则。但在机器学习系统中，遗传算法一般不采用下一代染色体整个取代上一代染色体的做法，而是采用部分重叠的方法，以保持系统性能的稳定。

竞争性的信用分配和以遗传算法为核心的规则发现构成了基于分类器的机器学习系统的基础，下面将对规则和消息系统、桶队算法及遗传算法作详细介绍。

4.2.1 规则与消息

图4.1是分类器系统的一般结构。消息经探测器进入分类器系统后，分解成一个或若干有限长度的消息再存入容量有限的消息表。消息表中的消息可以用来激活被称为分类器的串规则，被激活的分类器也将其消息部分发送到该表中，这些消息又可激活其它的分类器或触发系统的效应器执行某个环境动作。因此，在分类器系统中，分类器协调着从外部环境的输入的接收、处理到触发环境动作的执行

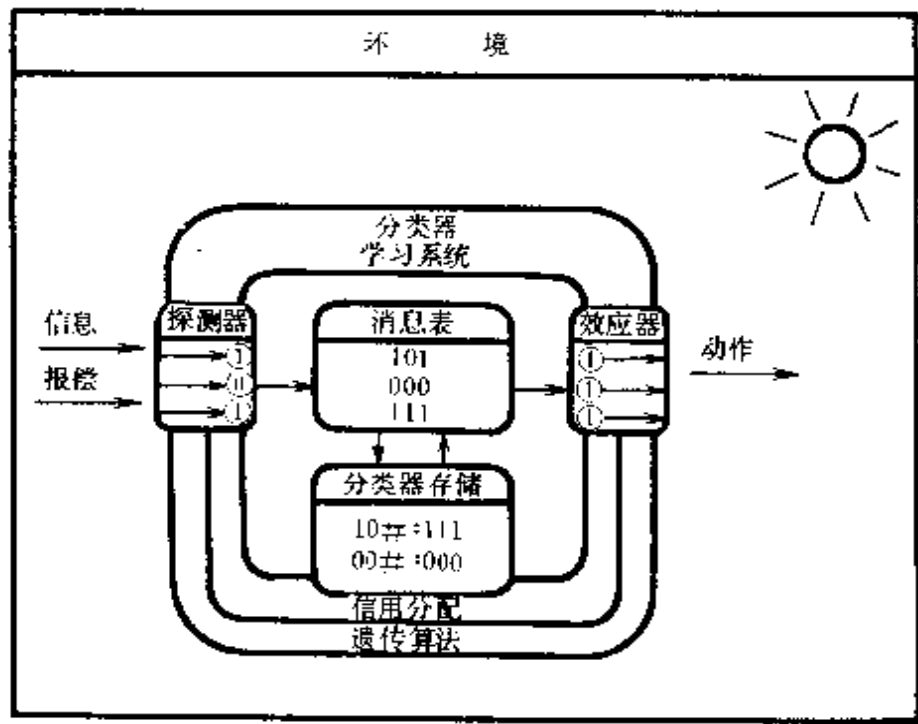


图 4.1 分类器系统的一般结构

这一全部过程。

分类器系统中,消息的长度都是固定的,每位均取自某特定字符集。所有的规则都采取条件/动作形式,每个条件对应满足相应规则的消息,每个动作对应了规则的条件被消息表中某个消息满足时该规则所要发送的消息。为便于遗传算法的实现,消息的可能取值都限定为二进制数字串。消息、条件及分类器的定义如下:

$\langle \text{message} \rangle ::= \langle m_1, m_2, \dots, m_j, \dots, m_k \rangle, m_j \in \{1, 0\}$

$\langle \text{condition} \rangle ::= \langle s_1, s_2, \dots, s_j, \dots, s_k \rangle, s_j \in \{1, 0, \# \}$

$\langle \text{classifier} \rangle ::= \langle \text{condition} \rangle / \langle \text{message} \rangle$

其中符号“#”为通配符,可以匹配“0”与“1”。通过这种方法,可以方便地将分类器的条件部分用长度为 k 的 0, 1, # 串表示出来,而且,可以通过适当的分类器组合,实现分类器条件的“AND”,“OR”及“NOT”操作。由 AND 操作连接的若干条件可表示成一多条件分类器。如 $M1 \text{ AND } M2$ 可写成 $M1, M2/M$; $M1/M$ 和 $M2/M$ 这两个分类器表示成一个 OR 条件。当 $M1$ 或 $M2$ 出现在消息表中, M 就被发送到表中。非操作符可以被转换成一条件为 $\neg M$ 的分类器。

下面,我们通过一个关于视觉应用的简单分类器系统^[14,15]来说明规则与消息系统的运行情况。该分类器系统如图 4.2 所示,输入界面的一组探测器为视觉场中每个目标生成一消息,每个探测器对应目标的特定属性。该系统还有三种效应器,以确定它在环境中所要采取的动作。其中一个效应器控制视觉向量,该向量表示视觉系统相对视觉场中心的方向,每个时间步里,它都可以作增量式旋转,如旋转至视觉场中心的左方、右方或旋转 15 度等。第二种效应器控制运动向量的旋转,该运动向量与视觉方向是独立的。第三种效应器控制系统在指定运动方向

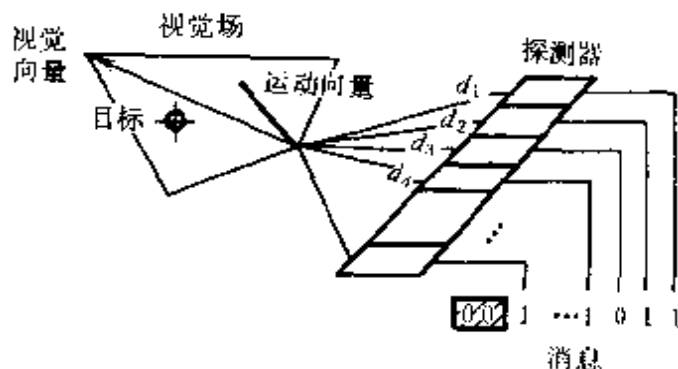


图 4.2 简单视觉分类器系统

上的速率,如快、慢、停等。分类器对探测器获得的输入进行处理,并将结果提供给效应器,以命令方式控制系统达到目的。图4.2中给出了用以确定来自输入界面最右边6个消息位的探测器。这些探测器根据相应属性输出二进制信息即环境消息。这6个探测器定义如下:

$$d_1 = \begin{cases} 1 & \text{若目标处于运动状态} \\ 0 & \text{否则} \end{cases}$$

$$(d_2, d_3) = \begin{cases} (0,0) & \text{若目标处在视觉场中心} \\ (1,0) & \text{若目标处在视觉场左侧} \\ (0,1) & \text{若目标在视觉场右侧} \end{cases}$$

$$d_4 = \begin{cases} 1 & \text{若视觉系统与目标邻近} \\ 0 & \text{否则} \end{cases}$$

$$d_5 = \begin{cases} 1 & \text{若目标较大} \\ 0 & \text{否则} \end{cases}$$

$$d_6 = \begin{cases} 1 & \text{若目标有斜纹} \\ 0 & \text{否则} \end{cases}$$

按照这些探测器的定义,我们将下面这条规则转换成对应的分类器形式:

if there is "prey" (small, moving, nonstriped object)
 centered in the vision field (cencered) and,
 not adjcent(nonadjacent)
 then move toward the object (ALIGN) rapidly(FAST)

为实现这种转换,首先,将规则的条件部分用0,1及#表示出来。其中“#”对应的位表示其取值如何不影响该规则的好坏。此外,由输入界面生成的消息的最左两位为标志位,规定为(0,0)。这样,上面给出的那条规则对应的分类器的条件可以表示为如下形式:

00## ## ## ## ## ## ## 000001

假设该条件满足时,分类器发送出如下的消息:

0100000000000000

其中前缀“01”表示该消息并非来自输入界面。如果进一步假设该消息被直接用作输出界面效应器的触发条件,则上面的那条规则对应的分类器的完整形式为:

00#####000001/0100000000000000,ALIGN,FAST

4.2.2 桶队算法

在分类器系统中,如何评价各分类器优劣是一个关键问题。在众多的评价方法中,最有名的是 Holland 提出的桶队算法^[4]。

对学习系统来说,其核心是执行系统,因为学习系统试图改进的正是执行系统的动作。与执行系统有关的一个重要问题就是如何在复杂的执行环境中对导致某些最终结果的单个决定(例如在下棋中,是对胜负起决定作用的单个棋子移动)分配信用。这是一个非常复杂的归纳任务,对于分类器系统来说,它必须确定出哪些规则对效应器的触发起着重要作用,以及这些规则的条件是否具有一定的代表性,即规则的激活频率是否较高。由于在分类器系统中,有可能多个分类器同时被激活,而且被激活的分类器的作用不一定立即显现出来,而是要通过一系列的分类器的激活、消息发送的过程才能确定。所以信用分配是分类器系统中一个困难任务。

如前所述,目前最重要的实现信用分配任务的方法是 Holland 的桶队算法。这是一种分步式的增量信用分配模式,其思想源自经济领域。在该模式下,全体分类器都视为处在一信息经济环境中。就如同商品经济中,从信息生产者到最终消费者之间,由分类器构成一条链,链上的分类器可视为中间商。它只与其“供货者”(即发送出的消息满足其条件的规则)和“消费者”(即与发送出的消息所匹配的那些分类器)直接相关。信息通过链来实现其传送过程,开始由探测器启动,中间由分类器通过激活和消息发送来延续信息的传送,直到效应器被触发而执行某个动作为止。实际上,这一过程较复杂,有两个问题必须要解决:一是当多个规则同时都要求被激活时,如何解决竞争

问题；二是对一规则被激活产生过作用的那些规则如何分配信用。

为解决上述两个问题，分类器系统中的信用分配算法借用了经济领域的思想。其中拍卖行和票据交换所就是从类比而来的两个重要概念。引入拍卖行后，获得匹配的分类器并不立即发送其消息，而仅是获得参与一次激活拍卖的资格。为参与拍卖，每个分类器必须记录其强度 S （亦称净值），而且要出一个与其强度成正比的叫价 B 。这样，在拍卖活动中，强度大的规则就可能获胜而被获准向消息表发送消息，并通过票据交换所结算，即将其叫价提供给激活该分类器的分类器。如果该分类器所发送出的消息也能在后续的匹配周期中激活其它分类器，则它同样也能从被它激活的分类器上获得叫价的一部分或全部，因而强度也会得到增加。

从这种过程来看，一条规则若要获利（即强度增加），那么它的消费者也必须从下一个消费者上获利。这样构成的消费者链将结束于若干最终消费者，即达到目标并从环境直接获得报偿的规则。如此，该链上的所有规则都能得到报偿。但若某个规则序列导致一错误结论，则序列上的最后一条规则的强度将减小，并且沿着序列回溯，若回溯点对应的规则在新一轮叫价低于与其竞争的其它规则，则链将沿其它方向继续扩展。

有的叫价方法还考虑了分类器的条件的特殊性。例如，在 t 时刻消息 01010 与两个分类器 0# #1# /11001 和 01# 10/11001 的条件均匹配，且它们的强度都是 200，叫价系数为 0.1，则它们的叫价都是 20。此时，我们可能倾向于激活条件部分更特殊的分类器。为反映出这种倾向性，有的叫价方法中取叫价为叫价系数、分类器强度以及分类器条件中非通配符（即 0,1）数三者之积。也就是 $B_i(t) = C * R_i(t) * S_i(t)$ ，其中 $B_i(t)$ 表示 t 时刻分类器 i 的叫价， C 为叫价系数， $S_i(t)$ 为分类器 i 在 t 时刻的强度， $R_i(t)$ 为 t 时刻分类器 i 的条件部分 0,1 数。

桶队算法的工作原理可以借助图 4.3 来说明。假定 4 个分类器的初始强度均为 200，向系统发送的初始环境消息为 0111，叫价系数为 0.1，叫价为叫价系数 $C_{b,d}$ 与强度之积。 $t=0$ 时，叫价为 20 的分类器 1 被

匹配,获准在下一时间步(即 $t=1$ 时刻)发送消息。分类器1将其叫价支付给激活它的环境,因而环境的强度增加20。在后续的时间步里,被激活的分类器将其叫价支付给上一时间步里被激活的分类器。在第5时间步,系统从环境中得到报偿,并将其支付给最后一个被激活的分类器4。

我们可以用下面这样一个公式来描述第 i 个分类器的强度 S_i 被激活之后的变化:

$$S_i(t+1) = S_i(t) - B_i(t) - T_i(t) + R_i(t)$$

它表示第 i 个分类器在 t 时刻因叫价竞争获胜而被允许向消息表发送消息,并通过票据交换所支付叫价即 $B_i(t)$;同时该分类器也可能因以前的消息发送活动而得到其它分类器或环境的报偿 $R_i(t)$;此外分类器可能要向系统支付一定的税收 $T_i(t)$ 。 $T_i(t)$ 的确定有多种方法。如选择 $T_i(t) = C_{tax} * S_i(t)$,其中 C_{tax} 为一系数。在图4.3的

| 索引号 | 分类器 | $t=0$ | | | | $t=1$ | | | | $t=2$ | | | |
|-----|-----------------|-------|------|----|----|-------|------|----|----|-------|------|----|----|
| | | 强度 | 消息 | 匹配 | 叫价 | 强度 | 消息 | 匹配 | 叫价 | 强度 | 消息 | 匹配 | 叫价 |
| 1) | 01 11 11 : 0000 | 200 | | E | 20 | 180 | 0000 | | | 220 | | | |
| 2) | 00 11 01 : 1100 | 200 | | | | 200 | | 1 | 20 | 180 | 1100 | | |
| 3) | 11 11 11 : 1000 | 200 | | | | 200 | | | | 200 | | 2 | 20 |
| 4) | 11 11 00 : 0001 | 200 | | | | 200 | | 1 | 20 | 180 | 0001 | 2 | 18 |
| 环 境 | | 0 | 0111 | | | 20 | | | | 20 | | | |

| 索引号 | 分类器 | $t=3$ | | | | $t=4$ | | | | $t=5$ | 报偿 |
|-----|-----------------|-------|------|----|----|-------|------|----|----|-------|----|
| | | 强度 | 消息 | 匹配 | 叫价 | 强度 | 消息 | 匹配 | 叫价 | 强度 | |
| 1) | 01 11 11 : 0000 | 220 | | | | 220 | | | | 220 | |
| 2) | 00 11 01 : 1100 | 218 | | | | 218 | | | | 208 | |
| 3) | 11 11 11 : 1000 | 180 | 1000 | | | 196 | | | | 196 | |
| 4) | 11 11 00 : 0001 | 162 | 0001 | 3 | 16 | 146 | 0001 | | | 206 | 50 |
| 环 境 | | 20 | | | | 20 | | | | 20 | |

注: 1. $C_{BID} = 0.1$

2. $C_{TAX} = 0.0$

图 4.3 桶队算法执行过程

例子中,选择了 $C_{\text{tax}} = 0$ 。

4.2.3 遗传算法

桶队算法有效地解决了规则评价及规则竞争的问题,但还有另外一个重要问题必须解决,即在系统对分类器作出评价之后,如何用新的分类器来取代强度低的分类器,以使分类器系统的整体性能不断提高。为此,分类器系统采用了遗传算法来实现这一目标。每个分类器对应一个染色体,全部分类器构成一个染色体集合,再通过遗传操作算子生成新规则,并利用它们取代原规则集中强度低的那些规则。一般来说,为保证学习系统性能的稳定性,不采用完全取代的方法,而是选取一定比例的染色体来取代。另外,在分类器系统中何时调用遗传算法来产生新规则也需要一个参数来控制。一种做法就是引入一个参数 T_{ga} ,它表示两次遗传算法调用的间隔时间(规则与消息周期数)。这样,每隔 T_{ga} 时间,就调用一次遗传算法,或者是以平均时间间隔来随机地调用遗传算法。有时,某些特殊的情况也可引发对遗传算法的调用,如所有消息分类器的条件都未能匹配,或者是系统的性能太差等。

根据积木块假定,遗传算法是通过不断地搜索并组合有效积木块来获得最优或近似最优解的,因此,下面我们将从积木块角度来介绍遗传算法在分类器系统中的应用。

在4.2.1节中,我们曾将分类器的条件部分定义为一长度为 k 的字符串 $\{0, 1, \#\}^k, j = 1, 2 \dots k$ 。我们可以将字符串的某一子部分,例如01或 $\#010$,看成一积木块。包含该积木块的全部染色体可以通过引入一个特殊符号“*” (表示对应位与该积木块无关)来定义。也称它为模式。例如, $*1\#\#0* * * \dots *$ 是一个只关心第2位到第5个位置上为 $1\#\#0$ 的一个模式,它定义了相应位置为积木块 $1\#\#0$ 的一个条件集。一般地,长度为 k 的染色体所包含的模式集为 $\{1, 0, \#, * \}^k$, 其中每个模式定义了所有可能条件的集合的一个子集,而每个条件又相应地定义了所有可能消息的集合的子集。

通常,对于一个给定模式 b ,在某时间步 t ,分类器系统中都包含很多该模式的分类器,即系统中有很多 b 的实例。由此,我们根据 b 的这些实例的强度来给 b 一个强度值 $V(b)$ 。假设 $t=i$ 时,模式 b 有 m 个实例 C_1, C_2, \dots, C_m , 则 $V_i(b) = S_i(C_j)/m$, 例如 $t=i$ 时, $b = *0 \# \# 1 * * \dots *$, 实例 C_1, C_2 的条件部分分别为 $10 \# 110 \dots 0$ 及 $00 \# 1011 \dots 1$, $S_i(C_1) = 4, S_i(C_2) = 2$, 则 $V_i(b) = (S_i(C_1) + S_i(C_2))/2 = 3$ 。

下面是 Holland 给出的在分类器中采用遗传算法的核心步骤:

- (1) 根据分类器的强度从分类器集合中成对挑选分类器, 强度越大, 被选出的可能性越大;
- (2) 对选中的分类器对应用交叉算子, 生成新的分类器;
- (3) 用生成的后代替换强度最弱的那些分类器。

从直观上看, 若模式 b 对应的所有实例的平均适应度 $V_i(b)$ 高于全体分类器的平均适应度 V_i , 则可以认为 b 是构造新的分类器的一个有效模式。遗传算法的优点就在于它能够迅速地搜索到并利用这些有效模式。为更好地引导遗传算法对有效模式的搜索和保存, 基于上述的直观思想, 分类器系统采用一种启发式信息, 即 $V_i(b)/V_i$, 但遗传算法实际上不可能直接利用它, 因为分类器条件包含 k 个字符, 可能的模式为 2^k 。这样, 集合 $\{V_i(b)/V_i\}$ 太大, 但我们可以间接地运用这一启发式信息, 其相应的遗传算法的主要步骤如下:

算法4.1 分类器系统中的遗传算法

begin

- (1) $t=0$, 随机生成集合 B_t , 它由 M 个分类器组成;
- (2) 计算 B_t 中全体分类器的平均强度 V_t , 对每个分类器赋予一个标准化强度值 $S_t(C_j)/V_t$;
- (3) 给 B_t 中的每个分类器 C_j 赋予一个与其标准强度值成正比的概率, 并根据 B_t 中的概率分布, 从 B_t 中选取 n 个分类器, 其中 $n \ll M$;
- (4) 对每对分类器应用交叉算子, 生成 $2n$ 个新的分类器;

(5) 将 B_t 中的 $2n$ 个强度值最低的分类器用新生成的 $2n$ 个取代;

(6) $t=t+1$, 返回步骤2。

end

下面, 我们通过一个简单的例子来说明上述算法。如图4.4所示, 假设 B_t 中有8个分类器。为便于描述, 本例仅给出分类器的条件部分:

| | | |
|-------|---------------|----------------|
| C_1 | 111011101 ... | $\leftarrow 1$ |
| C_2 | 110011110 ... | $\leftarrow 0$ |
| C_3 | 000111110 ... | $\leftarrow 2$ |
| C_4 | 011010011 ... | $\leftarrow 0$ |
| C_5 | 100011101 ... | $\leftarrow 2$ |
| C_6 | 011001100 ... | $\leftarrow 2$ |
| C_7 | 100100101 ... | $\leftarrow 0$ |
| C_8 | 100010010 ... | $\leftarrow 1$ |

其中箭头右边的值为分类器的强度。从上面的例子可知, C_2, C_3, C_6 是模式 $1 * * * * 11 * 0 * \dots * b_1$ 的实例, C_3, C_5, C_8 是模式 $* 00 * 1 * * \dots * b_2$ 的实例。对每个模式 b_1, b_2 计算其平均强度如下:

$$V(b_1) = (0 + 2 + 2)/3 = 1.33$$

$$V(b_2) = (2 + 2 + 1)/3 = 1.67$$

从图4.4可知, 在经过遗传操作之后, C_2, C_3, C_6 是模式 $* * * * * 1 * \#$ 的实例, 其平均强度为

$$V(* * * * * 1 * \#) = (0 + 2 + 2)/3 = 1.33$$

C_3, C_5, C_8 是模式 $* 00 * \# * * * * *$ 的实例, 其平均强度为

$$V(* 00 * \# * * * * *) = (2 + 2 + 1)/3 = 1.67$$

通过该例子可以看出调用一次遗传算法后, 新生成的若干分类器取代了父代中较弱的那些分类器。但有时需要解决这样一种问题, 即当染色体集合中各染色体之间有太多的相似性时, 要采取某种机

制,如在替换时将非常相似的染色体用新生的染色体取代,以避免染色体集合收敛到一个局部最优解上。

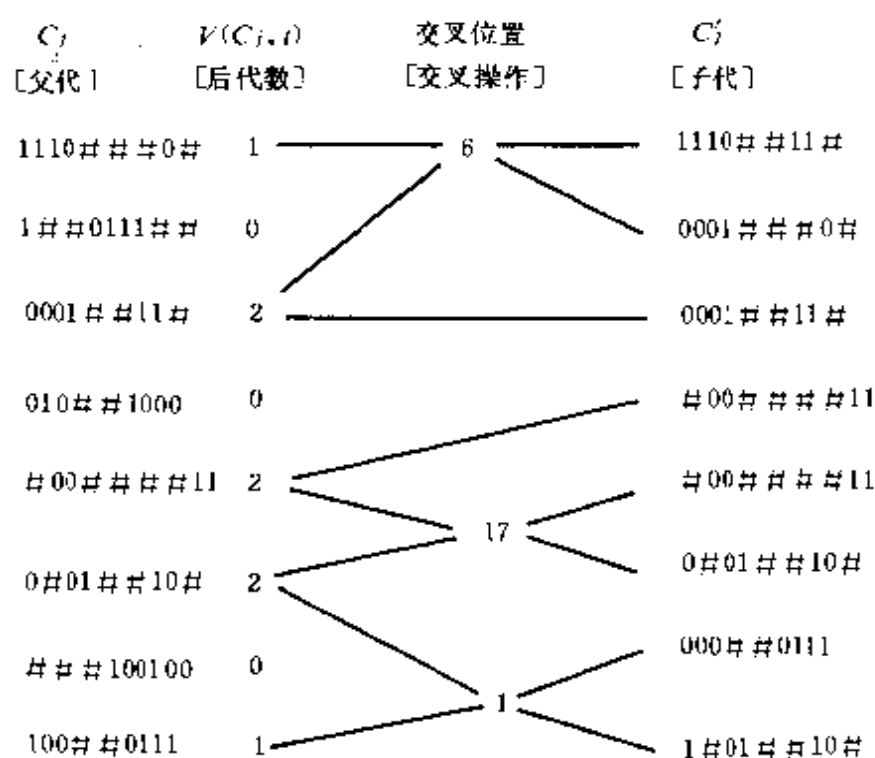


图 4.4 交叉操作

4.3 学习系统 LS-1^[13]

4.3.1 LS-1与 CS-1的区别

自从1978年 Holland 与 Reitman 实现了第一个分类器系统 CS-1以后,关于 GA 在机器学习中的应用研究受到了人们的重视,并由此而提出和实现了相应的系统。其中最著名的是 Smith 的学习系统 LS-1^[5]。尽管 LS-1诞生在 CS-1之后,但 LS-1系统在若干重要的方面与 CS-1有根本性的差别。具体表现在字符串规则、染色体表示方法、搜索结构的形成以及遗传操作算子的应用上。

CS-1与 LS-1的主要不同可以用图4.5来说明。其中图4.5(a)给出了 CS-1的体系结构,通过它我们可以看到在 CS-1中,遗传操作对

象是单个的规则,在信用分配算法运行期间,通过信用分配机制来评价所有规则。在遗传算法运行周期,规则之间进行交叉等操作。与此相对应的,图4.5(b)显示了 LS-1的内在结构,在这里遗传操作与评价的对象与 CS-1相比已提升到整个规则集。

正是由于 LS-1的这种遗传操作对象级别的提升,使得 LS-1与 CS-1之间产生了根本性的差别。并因此而回避了信用分配问题,因为在 LS-1中要评价整个规则集的好坏,而不再是单个规则在整个规则集中的作用如何。当然,信用分配问题的回避也有其负作用,因为此时执行系统向学习系统的反馈大大减少,从而使得学习系统需要更多的测试才能获得令人满意的执行性能。

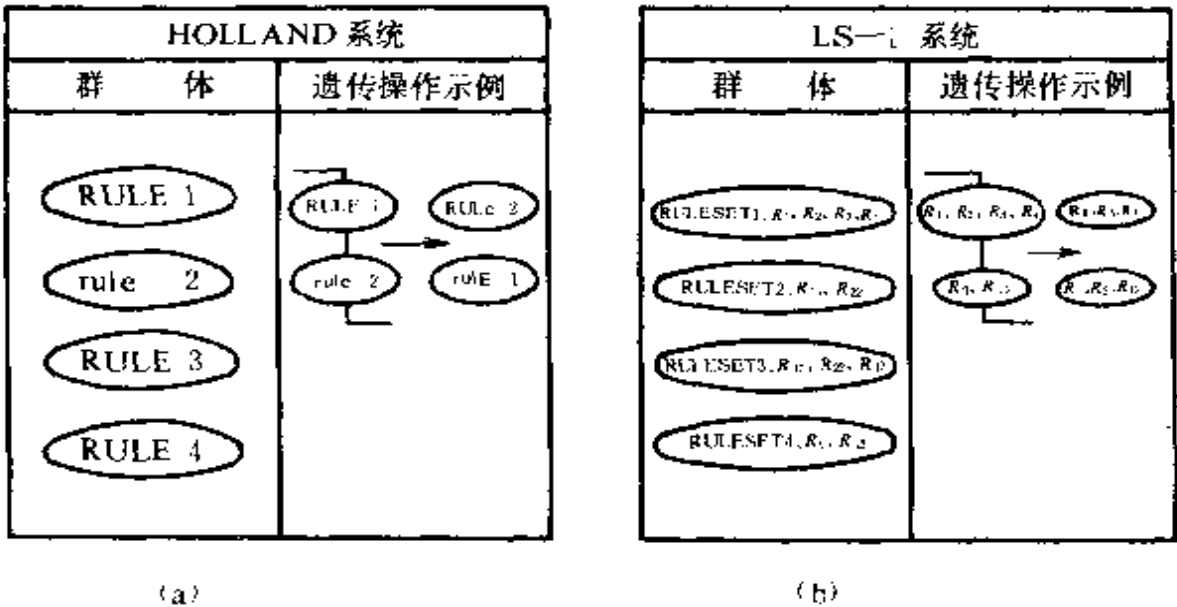


图 4.5 (a) 类 CS 系统 (b) 类 LS 系统

4.3.2 LS-1的工作原理

LS-1中包含了一个推理机的规则,它们是产生系统与分类器系统的一种有机的结合。该系统的工作内存中的内容构成一个无序的定长二元字符串集合。其中每个字符串分为信号部分与数据部分。产生式内存中存放的是一个无序的规则集,其中每条规则为一定长字符串,规则的条件由 k 位固定模式构成,前 i 位与 i 个环境探测器一一对应,其余的 $k-i$ 位模式对应着工作内存的信号部分。如同分类

器系统,LS-1采用的也是并行控制策略,即所有匹配的规则同时点火,除非有规则能引发外部动作的执行,此时它们就被标记,然后,再根据与各个外部动作对应的规则数成正比的概率,从中选择一个外部动作执行。

关于LS-1的匹配操作可以通过下面这个例子说明。假设该系统有两个环境探测器及一个工作内存。具体见图4.6。此外,还有如下的产生式规则:

— 1 # # 00 # # 0 # 1 # # x0 # # x001Y → 011 REASSERT(Y)

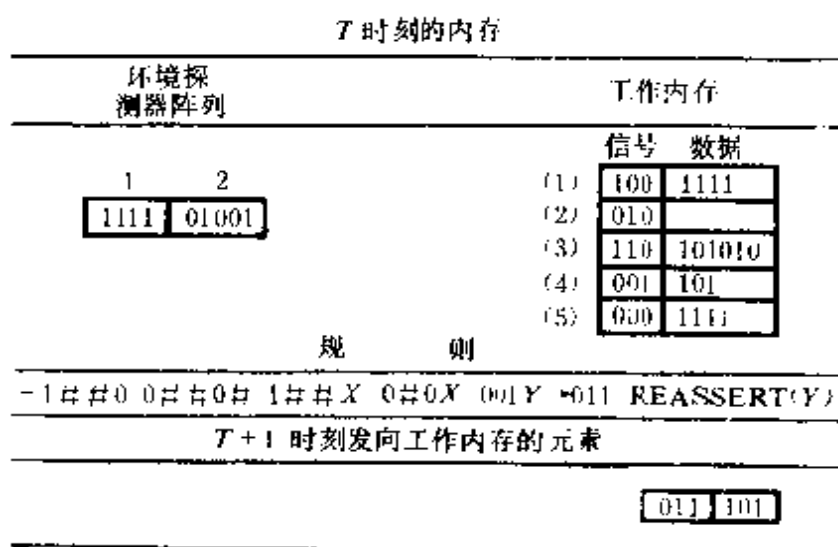


图 4.6 LS-1的工作原理

虽然它看起来有些像上一节介绍的分类器,但实际上它们有一些差别。首先这里规则的条件被分成两部分:其中一个环境部分,即 — 1 # # 00 # # 0 #;一个是工作内存部分,即条件的其余部分,它被划分成若干工作内存编组(working memory groupings)。其次,除了有0,1及#等字符外,还包含了其它的符号,其中第一个探测器组前的“—”表示逻辑“非”。如果某个环境消息或工作内存数据与带“—”的模式不匹配,则表示该模式被满足。在该例中,由于环境消息1111与模式1 # # 0不匹配,模式— 1 # # 0被满足。接着对规则向右继续扫描,第二个环境消息01001同第二个模式0 # # 0 #匹配,至此该规则的环境模式部分都已满足。然后再对工作内存进行扫描,检查规

则的工作内存编组部分是否能被匹配,LS-1规则的每个工作内存编组由前缀、模式及后缀组成,如同环境部分,每个工作内存编组的前缀可以是逻辑非(用“-”表示),还可以是忽略符号,该符号令解释器将编组忽略。工作内存编组的后缀可以是空、X、Y。其中X和Y是LS-1的两个合法变量名。若在匹配过程中,变量X和Y被扫描到,则工作内存中某一单元的信号部分与该变量关联的工作内存编组——模式匹配,且该变量被赋以工作内存中与信号部分在同一单元的数据,如在该例中,第一个单元信号100与规则中的第一个工作内存编组的模式1##匹配,从而使变量X取值为1111。第二个工作内存编组中的X也以1111代换。接着,模式0#0被第2个单元的信号部分匹配,然后,因为X已代换为1111,所以该模式与第2个工作内存编组是不完全匹配。接下来对工作内存扫描,发现第5个单元与之匹配,第4个单元的信号部分与第3个工作内存编组的模式匹配,因此Y被赋值101。

变量X和Y给了LS-1一种基本的识别数据元素之间等同性的能力。此外,它们也为系统提供一种将信息从规则左部传递至右部的能力。一旦一条规则的左部被完全匹配,系统将要执行下面的两个任务:首先,将其信号发送到某工作内存单元,假设为单元2;其次,规则将对其动作及动作参量进行评价,以确定发送到单元*i*的数据。就该例而言,一旦规则被匹配,其信号011被送至一空闲工作内存单元,并执行其动作REASSERT。动作REASSERT只是将变量Y的值101放入信号011所在单元的数据部分。这样,系统在工作内存中又占用一个单元并填入相应的信号与数据011101,规则右部的动作既可以是独立于任务的,如此例中的REASSERT,也可以与任务相关。

上述的规则结构和推理机制非常直接,其遗传操作也是如此。在LS-1中共有4种操作,它们是选择、变异、交叉和逆转。其中选择及变异操作与标准的遗传操作并无差别,但对交叉操作来说,因染色体是变长的,所以必须在标准的交叉操作基础上加以限制,以确保有意义的积木块被交换。在LS-1中,这样限制性交叉操作是通过以下几

个步骤实现的:1)调准;2)选择交叉点;3)交换。与标准的交叉操作相比,带限制的交叉操作多了调准这一步,其具体含义可通过一个例子来说明,该例子中有两个规则集 RS_1 和 RS_2 , $RS_1 = R_1;R_2;R_3$, $RS_2 = R_8;R_9;R_4;R_7;R_6;R_5$ 。其中 R_1 至 R_9 分别表示9条规则,冒号表示规则之间的分隔符。当执行调准操作时,首先要为 RS_1 和 RS_2 分别选择一个调准点,假设 RS_1 的调准点为 R_1 , RS_2 的调准点为 R_4 。然后,使 R_1 和 R_4 对准:

$$RS_1 = R_1;R_2;R_3$$

$$RS_2 = R_8;R_9;R_4;R_7;R_6;R_5$$

接下来,在两个规则集重叠区域内选择交叉点。交叉点可以在规则的边界上或者在规则的内部,最后进行交换。

逆转操作符在以前章节曾提及到,不过此处对逆转 Λ 的位置的

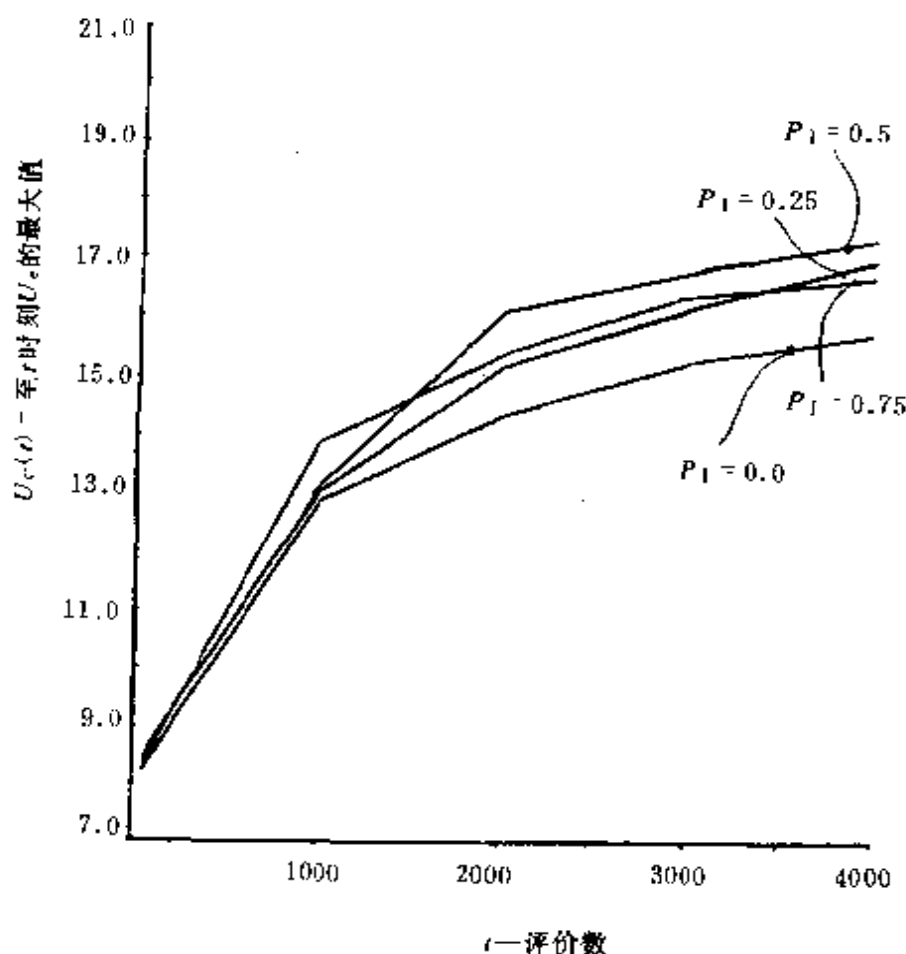


图 4.7 不同的逆转操作概率 P_1 对应的函数 $U_c(t)$ 的性能

选择有一点限制：它们必须在规则的边界上。例如，对 RS_2 作逆转操作，首先选择逆转点：

$$RS_2 = R_8 : R_9 : R_4 : R_7 : R_6 : R_5$$

$$\quad \quad \quad \wedge \quad \quad \quad \wedge$$

接着将逆转点之间的规则按其逆序重排，得到新的规则集：

$$RS_2 = R_8 : R_7 : R_4 : R_9 : R_6 : R_5$$

Smith 通过一个走迷宫 (maze running) 问题证实了逆转操作在 LS-1 中的作用，即它对学习系统的收敛性及收敛速度都有益处。图 4.7 是 Smith 给出的关于逆转操作作用的曲线图。

4.4 基于遗传算法的概念学习系统

从例子中进行概念学习是机器学习研究领域中最活跃的分支之一，其目标是从概念的特殊例子中归纳出概念的一般性描述，它是归纳学习的一种重要形式。归纳学习的一般性模式是从刻划某些对象、情况、过程等的一些特殊知识即实例集合 F 及一个临时的归纳假设及背景知识出发，在假设空间搜索一个强蕴含或弱蕴含实例并满足背景知识的一个归纳假设 H 。

由于对于任一给定的实例集合，都可能生成大量的蕴含这些实例的假设，因此需要提供背景知识加以约束和提供优选标准将大量的选择缩减到一个或几个最优的假设。定义这种标准的一种典型方法是确定出假设的选择特性，如要求假设是同所有实例一致且是最短的描述。

在概念学习中，若实例事先已被分类，则称为受指导的概念学习，它将从目标概念的一组正例和反例中归纳出概念的一般性描述。

一般地，实例被表示成 n 维特征向量，因此概念可视为该向量空间的若干点组成的子集。概念学习程序是从关于特征空间的描述及一组事先已分好类的例子出发，运用有关归纳操作来生成比较精确的目标概念描述。概念学习的困难之一是其无限制性，即对概念的描述

之详细程度和描述该概念可能采取的多种多样的形式之间不存在必然的限制。但在机器学习中,要非常仔细地定义出目标和要求解决的问题,它包括用何种形式表示概念、定义语言、定义推论、定义可能被使用的推理方式。所选择的描述语言应使得概念的重要特性容易编码,并忽略次要及无关的信息。因此,用以表达概念的语言类型可作为分类概念学习方法的指导标准。最常见的是产生式规则和决策树这两种形式,它们分别用各自不同的形式定义了一个概念描述空间。正确的概念描述必须通过正例及反例为约束才能从该空间中被搜索到。但由于搜索空间的庞大及复杂性,可能还需要其它的一些倾向知识(bias),如倾向于概念的描述更简单,更一般等。但是这些倾向性知识的添加只对与其匹配的那些概念的形成有良好效果,因此我们需要一种能从整体上提高概念学习程序的鲁棒性与适应性,即使在对所要学习的概念的先验知识知之甚少情况下,学习的结果也能令人接受的方法。遗传算法因在复杂的搜索空间中显示出良好的自适应性而被越来越多地应用到概念学习领域中。如前所述,概念学习中描述语言的选择对学习方法与效果有重要的作用。因此,在概念学习中应用遗传算法要既能反映出要解决的问题的本质特征,又能将可能的解表示出来。目前在基于遗传算法的概念学习中主要采用的是基于规则的形式表示,但由于学习系统在运行前无法知道所需规则的数目,所以传统的遗传算法表示方法对它是不合适的。在基于遗传算法的概念学习中,人们普遍采用的方法有两种:一种是 Michigan 方法;另一种是 Pittsburgh 方法。这两种方法的区别在于遗传算法在学习过程中发挥的作用不同。Michigan 方法的染色体集团中的每个成员即染色体不能单独构成解。在这种方法中,概念是由一个染色体集合构成。遗传算法的作用仅限于生成新规则,而生成的新规则如何插入到当前的模型则由其它机制去完成。与此不同的是 Pittsburgh 方法似乎更适合受指导的概念学习任务,因为染色体集团中的每个成员都代表了一个完备的候选概念,遗传算法的作用能够扩展至新规则的集成。本节将结合 Spears 和 De Jong 的 GABIL 系

统^[16]介绍基于遗传算法的概念学习的基本方法。

4.4.1 搜索空间的表示

为了将遗传算法应用到概念学习问题上,首先要选择一种有效的表示方法来描述搜索空间即可能的概念集合。这里有两种方法可供选择:一种是采用复杂的非字符串形式即直接的表示方法;另一种就是采用传统的字符串表示方法。这两种方法各有其优点,若采用与问题直接相关的表示形式,则遗传操作是针对该表示的语法及语义而特别设计的,因而可能特别有效,但所需的精确形式及相应的实施频率都是难以确定的。若采用字符串这种间接表示方法,则需要建立一种映射将复杂的概念描述转换成线性的字符串,以使标准的遗传算法能对其实施相应的操作。GABIL 采用的是第二种方法。通常复杂的概念可以很自然地用一个可能有部分重叠的分类规则的析取集来表达,也就是将概念表示成析取范式,每条规则的左部由若干关于特征值测试的合取式构成,规则的右部表示那些满足规则左部合取式的实例应归属的概念类。如果这样的规则集能够对特征空间中的实例正确分类,则它就代表了所要学习的概念。在 GABIL 中,每条合取规则都采用对应的二进制形式表示,其转换方法比较简单,每个染色体长度固定,它包含了对例子的全部特征的测试,而每种特征的测试又是由固定长度的二进制串组成,其长度依赖于特征的类型,如范畴性、线性、或结构性的等等。目前,GABIL 系统只考虑了具有范畴性质类型的特征。如果一个范畴型特征具有 k 种可能取值,则在染色体中就为其分配 k 位,每一位与一种特定值对应,其取值0或1与该值所关联的特征是否为规则条件部分的一个合取项有关。例如,下面这样一条规则:

IF ($F_1 = \text{large}$) and ($F_2 = \text{sphere or cube}$) THEN
it is a widget

它的左部是一个合取式。其中第2个或取项包含了内部析取式。假设我们仅依靠 F_1 及 F_2 这两个特征来学习 widget 及 gadget 这两个概念的描述, 且 F_1 的取值域为 {small, medium, large}, F_2 的取值域为 {spere, cube, brick, tube}, 概念 widget 用0表示, gadget 用1表示, 则上面的规则可以表示成如下的染色体形式:

| | | |
|-------|-------|-------|
| F_1 | F_2 | class |
| 001 | 1100 | 0 |

反过来, 染色体11110000对应规则为:

```
IF ( $F_1$  = small or medium or large) and ( $F_2$  = spere)
THEN
it is a widget
```

在该规则中, 特征 F_1 取值域中任何值, 对应的合取项都成立。这表示该特征在这条规则中是无关的。因此, 可以将相应的关于特征 F_1 取值的合取项舍弃。上述规则可简化为

```
IF ( $F_2$ =spere) THEN it is a widget
```

由于一个概念描述往往是多条规则组成的, 所以 GABIL 中染色体实际上表示的是一个规则集合, 而且染色体长度可各不相同, 即其代表的规则集合的大小不一样。例如, 下面两条规则组成的集合

```
IF ( $F_1$ =small) THEN it is a widget
or
```

```
IF ( ( $F_1$ =medium or large) and ( $F_2$ =brick) ) THEN it is
```

a widget

可表示成

| F_1 | F_2 | class | F_1 | F_2 | class |
|-------|-------|-------|-------|-------|-------|
| 100 | 11110 | | 011 | 0010 | 0 |

4.4.2 遗传操作

采用这种二进制串表示后,还必须定义一种简单的执行语义,并依赖这种执行语义定义相应的遗传操作。GABIL 的执行语义的特点是规则集中的规则无序,也就是说,在用规则集来预测某个实例属于哪一类时,对规则集中全部规则的左部都要进行检查,以确定该实例所属的类。然而,这样有可能使得某些例子被几条规则覆盖从而产生矛盾。解决这种矛盾的方法有多种,如对覆盖该例子的那些规则的复杂性进行度量,或采用各种选举模式。另一种可能的情况是一个例子可能不被规则集中的任何规则覆盖,其相应的解决方法是采用部分匹配或覆盖操作。这两种问题对于多概念学习任务来说都是很困难的,所以 GABIL 系统仅考虑单概念学习问题。如果一个实例被一个或多个规则所覆盖,则它就属于所学习的概念类的一个正例,否则就是一个反例。

GABIL 采用的变异操作与标准遗传算法相同,而交叉操作则在标准的两点交叉操作基础上作了一点限制,因为对标准的遗传算法来说,所有染色体长度相同,所以只要给定两个交叉点,要进行交叉操作的两个父代染色体各自交叉点之间的一段相互交换就可以了。但对变长染色体来说,每个父代染色体都必须给出两个交叉点,而且这两对交叉点的选择要满足染色体内在的语义约束。具体方法可以通过下面的例子来说明。第一个染色体的交叉点分别是第一条规则的第3和第7位上,第二个染色体的交叉点是第一条规则的第3位及第

二条规则的第7位上。

| F_1 | F_2 | class | F_1 | F_2 | class |
|-------|-------|-------|-------|-------|-------|
| 100 | 0100 | 0 | 011 | 0010 | 0 |
| 010 | 0001 | 0 | 110 | 0011 | 0 |

进行交叉操作后生成的下一代染色体如下,它们仍然是满足相应的语义要求的

| F_1 | F_2 | class | F_1 | F_2 | class | F_1 | F_2 | class |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 100 | 0001 | 0 | 110 | 0011 | 0 | 011 | 0010 | 0 |
| 010 | 0100 | 0 | | | | | | |

GABIL 系统在选择适应度函数时,采用了简化策略,它只关心每个染色体所对应的规则集的分类性能,而忽略了规则集的大小及规则的复杂性,其具体形式为:

$$\text{fitness} = (\text{percent correct})^2$$

这种形式的适应度函数为概念学习程序提供了一种倾向性,促进规则集相对于训练实例集的一致性 & 完备性,即使得规则集不覆盖反例且覆盖所有的正例。

4.4.3 执行过程

GABIL 系统中遗传算法的执行过程与标准的遗传算法相同,循环地执行选择、交叉及变异操作。但它们的循环是在搜索到一个一致且完备的规则集时结束。此外,调用遗传算法的方式与标准的遗传算法有所不同,它与学习程序在进行概念学习时所接受的训练实例的方式是直接相关的。一般说来,环境向学习程序提供的实例的方式分

批量式与增量式两种。对基于遗传算法的概念学习程序来说,采用批量方式要简单些,此时一个固定的训练实例集一次性地被提交给学习系统。然后,遗传算法开始运行,在规则集空间中搜索一个具有高适应度的规则集。若适应度为100%,则表示一个相对训练实例集完备且一致的规则集被发现。然而,在很多情况下,学习是一种无限的过程,新的实例是增量式获得的并向学习系统提供的,而且已提供的实例可能含有噪音。因此,概念学习系统必须具备从不具有代表性及含有噪声的实例中增量式地演化概念描述能力。

GABIL 系统中增量式学习是通过如下方式实现的:首先系统接收实例集中的一个实例,并调用相应的遗传算法在给定的资源限制下尽可能地搜索到一个一致且完备的规则集;然后系统接收下一个实例,并用已学习到的一个目前最优的规则集来对其分类。若分类错误,则将其与前一次得到的实例一起作为训练实例并再次调用遗传算法(此时可以认为是采用批量式学习)搜索覆盖这两个实例的规则集;若分类正确,则不调用遗传算法。依照上述方式,用新得到的规则集对下一个接收到的实例再分类。根据分类正确与否,重复以上过程。所以 GABIL 系统的学习方式可以看成是批量与增量的混合方式。在这种学习方式中,每个实例既是测试实例又是训练实例。

4.4.4 非标准操作

上面我们介绍了包含“纯”GA 作为其搜索机制的 GABIL 系统的内部表示,遗传操作及其学习方式。然而,尽管它的总体学习效果比较好,但它在某些方面却比其它一些系统,如 Utgoff 的 ID5R^[16]及 Iba 的 IACI^[17]差。其原因是 GABIL 系统中的 GA 太“纯”,它没将一些与概念学习任务有关的操作符融合进去。所以在扩展的 GABIL 系统中扩充了两种非标准的遗传操作符,即增加选择及降低条件。

增加选择项这一操作的目的是对学习到的概念作泛化处理。对一个概念来说,作泛化操作就是为它所包含的某些合取项增加一些

内部析取项。如 $(F_1 = \text{small}) \text{ and } (F_2 = \text{sphere or cube})$ 是对 $(F_1 = \text{small}) \text{ and } (F_2 = \text{sphere})$ 的泛化。在扩展的 GABIL 系统中, 这种操作(称 AA, adding alternative operation)是通过一种特殊的变异操作实现的。它与通常的变异操作区别在于 AA 的变异概率的非对称性。如某位变为 1 的概率取 0.75, 变为 0 的概率取 0.25。因此, 在施实了 AA 操作后, 内部析取项的增加可能性就比较大。当然, 同其它操作一样, AA 操作的施实也是有一定概率的。在 GABIL 的一个应用中, 该概率定为 1%。

另一种扩展的操作即降低条件是将概念条件部分的若干合取项从条件中去掉。例如, $(F_2 = \text{sphere})$ 是在对 $(F_1 = \text{small or medium}) \text{ and } (F_2 = \text{sphere})$ 作了降低条件操作后的结果。在 GABIL 系统中, 该操作称为 DC。当对某规则集(即一染色体)施实 DC 操作时, 每条规则的每个合取项都要进行检查以确定是否进行 DC 操作, 若一合取项中有半数以上的位取值为 1, 则将该合取项中其余为 0 的位变为 1, 其变换如下:

| F_1 | F_2 | 变为 | F_1 | F_2 |
|-------|-------|----|-------|-------|
| 110 | 100 | | 111 | 00 |

在 GABIL 的一个应用中, 该操作符的施实概率为 0.60。

4.4.5 GABIL 系统的自适应性

尽管遗传算法作为一种搜索机制具有很强的自适应能力, 但它所涉及到的诸多因素如染色体的大小, 染色体表示的类型, 操作类型及其概率的设置等在遗传算法的全部运行过程中都是静态的。为使遗传算法具有更强的自适应性, 人们提出了许多方法, 其目的都是将上述某些因素动态化。这些方法总括起来可以分为两大类型^[10]: 其中一类叫 within-problem 方法, 它考虑的是如何使遗传算法在求解某一问题的不同阶段自动地调整某些因素, 如操作类型、实施概率

等；另一类是 across-problem 方法，它关心的是如何解决遗传算法在求解不同的问题时作自适应性的改变。Grefenstette 曾于1986年提出元 GA 的思想^[18]，它是一个具有自适应能力的元遗传算法模块，该模块能够针对一类问题来自适应地调整标准遗传算法参数。此方法的优点是这样的系统对一类问题的解决都表现出很强的鲁棒性。但其缺点也较明显，其一是系统效率较低，对该类的每个问题进行求解，系统都必须运行多次才能获得较好的结果，亦即系统获得自适应能力的过程较为缓慢；其二是系统的这种自适应能力相对来说还是比较粗糙的，因为通过这种方式获得的自适应性对每个问题来说并不一定是最优的。

GABIL 系统的自适应能力是通过 within-problem 方法实现的，因而这种自适应是针对概念学习这一特定任务而表现出来的。在本节的前一部分内容曾提到扩展的 GABIL 系统中增加了两种与概念学习任务相关的特殊操作 DC 及 AA。对每个染色体来说，在遗传算法的全部运行过程中，这两种操作都是以固定的概率执行的。为了改变这两种操作执行上的静态性，自适应的 GABIL 系统在每个染色体上增加了两个控制位，每个控制位取0或1，其中一位控制 DC 操作，另一位控制 AA 操作。若控制位取1，且该位对应的是 DC 操作，则表示允许 DC 操作以预先设定的概率应用到相应的染色体上，否则根本不允许对相应染色体进行操作，相当于使概率为0。下面是增加了两个控制位的染色体例子：

| | | | | | | | |
|----------------|----------------|-------|----------------|----------------|-------|---|---|
| F ₁ | F ₂ | class | F ₁ | F ₂ | class | D | A |
| 010 | 001 | 0 | 110 | 011 | 0 | 1 | 0 |

正是这种小小的扩充，使得 GABIL 系统获得了同时对两种不同空间进行搜索的能力：一种是对概念空间的搜索以求最优概念；另一种是对操作空间的搜索以求最优操作集。当然，这里的操作空间仅是两种特殊操作的组合。

4.5 小 结

从早期的 Holland 的分类器系统 CS-1 和 Smith 的 LS-1, 到 Booker 的二维空间中的发现食物及避开毒物的分类器系统, Wilson 的 EYE-EYE 和 ANIMAT 等, 不仅对基于遗传算法的机器学习, 而且对遗传算法本身的发展都产生过不可忽视的作用。分类器系统在基于遗传算法的机器学习研究中影响较大, 但具体实现方法与要解决的问题有关。4.2 节介绍的是分类器系统的一般性结构, 它由规则与消息子系统、基于桶队算法的信用分配子系统及遗传算法三个主要部分组成。在这种分类器系统中, 染色体采用定长表示方法, 问题的解是由群体中若干染色体组成的集合。另一方面, Smith 的 LS-1 代表的是另一类方法, 每个染色体均表示问题的一候选解, 它由若干条规则组成, 由于不同的候选解中规则数无法事先确定, 因此, 染色体的长度各不相同。LS-1 的另一个重要特点是它完全回避了信用分配问题, 因为在 LS-1 中反馈是实时的。分类器系统对顺序决策问题的解决比较有效。基于遗传算法的概念学习是近几年来机器学习领域的一个较为引人注目的研究方向, 由于概念学习隐含的搜索机制, 使得遗传算法在概念学习中有用武之地。本章在 4.4 节通过 De Jong 的 GABIL 系统介绍了这种方法的基本实现手段, 目前也有一些嵌入领域知识的基于遗传算法的机器学习的研究, 如将概念学习中特有的操作遗传操作化, 并显示出一定的优点^[11]。此外, 学习分类系统的并行实现在基于遗传算法的机器学习研究中也占有相当份量, 有关此方面的内容请参阅本书的第五章。

参 考 文 献

- [1] Quinlan J. Induction of Decision Trees. Machine Learning, 1986,1(1):81~106
- [2] Michalski R, Mozetic L, Hong J, Lavrac N. The AQ15 Inductive Learning System: An Overview and Experiments (Technical Report No. UIUCDCS-R-86-1260). Urbana-Champaign, IL: University of Illinois, 1986.
- [3] Holland J H, Reitman. J. S, Cognitive Systems Based on Adaptive Algorithms. In: Waterman D A, Hayes-Roth F (Eds.). Pattern Directed Inference Systems, New York: Academic Press, 1978, 313~329
- [4] Holland J H. Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: Michalski R S, Carbonell J G, Mitchell T M (Eds.), Machine Learning II, Los Altos, CA: Morgan Kaufmann, 1986, 593—623
- [5] Smith S F. Flexible Learning of Problem Solving Heuristics Through Adaptive Search. Proceedings of the 8th International Joint Conference on Artificial Intelligence, 1983, 422~425
- [6] Booker L B. Intelligent Behavior as an Adaptation to the Task Environment (Doctoral Dissertation, University of Michigan). Dissertations Abstracts International, Vol. 43, No. 2, 469B, 1982.
- [7] Wilson S W. Adaptive "Cortical" Pattern Recognition. Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985, 16~23
- [8] Wilson S W. Knowledge Growth in an Artificial Animal. Pro-

- ceedings of an International Conference on Genetic Algorithms and their Applications, 1985, 88~196
- [9] Goldberg. D E. Computer—aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning (Doctoral Dissertation, University of Michigan). Dissertations Abstracts International, Vol. 44, No. 10, 3174B, 1983
 - [10] DeJong K A, Spears W M, Gordon D F. Using Genetic Algorithms for Concept Learning. Machine Learning, Vol. 13, Nos. 2/3, 1993, 161~188
 - [11] Janikow C Z. A knowledge—Intensive Genetic Algorithm for Supervised Learning. Machine Learning, Vol. 13, Nos. 2/3, 1993, 189~228
 - [12] Green D P, Smith S F. Competition—Based Induction of Decision Models from Examples. Machine Learning, Vol. 13, Nos. 2/3, 19
 - [13] Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. New York; Addison-Wesley, 1989.
 - [14] Booker L B, Goldberg D E, Holland J H. Classifier Systems and Genetic Algorithms. Artificial Intelligence, 1989, 40(1); 235~282
 - [15] ShiZ Z. Principles of Machine Learning. International Academic Publishers, 1992
 - [16] Utgoff P. ID5R: An Incremental ID3. Proceedings of 5th International Conference on Machine Learning, 1988, 107~120
 - [17] Iba G. Learning Disjunctive Concepts from Examples (A. I. Memo 548). Cambridge, MA; Massachusetts Institute of Technology, 1979
 - [18] Grefenstette J J. Optimization of Control Parameters for Genetic Algorithms. IEEE Transactions on Systems, Man, and

Cybernetics, 1986,16(1):122~128

第五章 遗传算法与并行处理

遗传算法固有的并行性和大规模并行机的快速发展,促使许多研究者开始研究遗传算法的并行化问题,研制数量更加接近自然物种的软件群体将成为可能。遗传算法与并行计算机的结合,能把并行机的高速性和遗传算法固有的并行性两者的长处彼此结合起来,从而也促进了并行遗传算法的研究与发展。本章首先从遗传算法的固有并行性及其并行化实现的困难讲起,探讨遗传算法的并行化途径,详细地讨论粗粒度孤岛模型和细粒度邻域模型的原理、研究现状和在并行机上的具体实现实例,并在此基础上对上述两种模型的性能作简要的比较,最后还对学习分类系统的并行实现作了简介。

5.1 遗传算法固有的并行性及其并行化的困难

5.1.1 源于自然的并行性

众所周知,在自然进化过程的任何时刻,总是同时有大量的物种在彼此独立地向前进化(当然,不同物种间的相互竞争使它们的进化过程会有不同程度的相互影响)。在同一物种内部,也是同时存在着大量的个体在通过自然选择、交配和基因突变而进化着。显然,自然界的进化过程本身就是一个并行过程。

遗传算法是人们对自然进化过程的机器模拟,它分别使用选择(select)、交叉(crossover)和变异(mutate)等遗传操作来模拟自然进化过程中的自然选择、交配和基因突变以达到求解组合搜索、优化和机器学习等问题的目的。遗传算法来源于自然进化,与自然进化有密不可分的关系,很自然地也就继承了自然进化过程所固有的并行

性。在遗传算法理论的形成过程中也充分地体现了这一点。

5.1.2 遗传算法理论中的并行性

遗传算法的奠基人 Holland 在最早提出遗传算法的理论和模型时就阐述了它所包含的固有的并行性^[1]。

Holland 最初所提出的适应系统的逻辑理论是后来明确形成的遗传算法的理论基础,该理论的模型可大致分为两大类:完全模型和非完全模型。其中,完全模型是指人工系统,这些系统具备某些从一开始就被完全规定好了的性能和规范,就像人们玩游戏时预先规定的游戏规则一样。完全模型中有一部分是在各种各样的并行计算机上实现的。非完全模型是指自然系统而言的。任何一个自然系统都包含有无限多的可变因素,而人们的理论不可避免地只能处理其中有限的为数不多的几个。正因为在自然系统中总是有一些变化因素,它们在人工理论中无法找到其清楚明确的对应成分,所以我们从理论中得出的结论必定是和自然系统大致相关的。而自然系统是一种并行系统,因此如果能够将我们的结论作些修改,使之能够适用于并行计算机的特定的编程体制,则对我们将是很有帮助的。这样,完全模型就证明了理论的正确性。同时,我们可以给每个完全模型找到与其相类似的一组非完全模型,这样,只要有合适的解释,一个给定的完全模型的结果就可以应用于与其相类似的一组非完全模型。这就是遗传算法能用于求解一大批实际问题的理论依据,同时它也提出了对遗传算法的并行化要求。

Holland 认为,对适应系统的研究的目的是为了发现系统该如何产生一些过程,使之能够进行有效的调整,从而去适应环境。适应系统应当被看作是由一些相互关联着的产生过程类,其中每一类过程都是适应系统所固有的。每个产生过程都联系着它所产生的一個程序代,适应系统对产生过程的任何修改都会引起整个程序代的变化。在遗传算法这样一个适应系统中,选择、交叉和变异等遗传操作就是这样一些产生过程,适应度函数则代表着环境,某个遗传操作作

用于某个个体就是产生过程所生成的一个程序。如果遗传算法中的某个遗传操作有所改变(如交叉策略有所变化),则整个群体的进化都将发生相应的变化。

显然,我们没有任何理由去限制产生过程在任何时刻只能产生一个程序。实际上如果让产生过程进行并行操作,即任何时刻都是产生一代而不是一个程序,那么系统将获得普遍的性能增益。同样,我们可以把环境也看作是同时对整个一代个体的要求。事实上,我们可以把适应系统逻辑理论中的对单个个体的操作都换成对整个一代个体的操作,在遗传算法中就是对一代个体同时进行适应度评估,同时进行遗传操作。上述就是包含在遗传算法理论中的固有的并行性。

5.1.3 遗传算法在并行实现上的困难

虽然遗传算法无论是从直观上还是理论上来看都具备并行性,遗传算法的并行化好像是势在必行、手到擒来的,其实如第一章所述的标准遗传算法在并行化的过程中会遇到通信量过大的困难。在并行实现中,同一代的个体被分布在若干个处理器上,根据全局适应度进行的选择操作以及作用于分布在不同处理器上的两个个体间的交叉操作等全局操作都会产生大量的通信开销,更何况这种通信是每代都有,造成的巨额开销只能导致整个系统效率极低,令人难以承受。这也是遗传算法发展至今,本身又具备固有的并行性,但在并行化方面始终进展甚少的原因。

目前,研究者们一致认为,必须对标准遗传算法进行改造,尽量减少巨量通信从而获得高效率。但是这么做如果使得遗传算法求解具体问题的质量有所下降,则岂不是得不偿失?这就是遗传算法在并行化中遇到的效率与效果之间的矛盾。任何对标准遗传算法的改造都必须以尽可能少地影响其进化效果为前提。

5.2 遗传算法的并行化途径

下面,从编程的角度来考虑遗传算法的并行化。首先让我们把一个遗传算法形式化描述如下:

```
begin
  (1) initialization
    (1.1)产生一个初始群体
    (1.2)评估第一代整个群体的适应度值
  (2) while running do
    (2.1)选择父代
    (2.2)交叉操作
    (2.3)子代变异
    (2.4)评估子代的适应度
    (2.5)子代取代父代,形成新的一代个体
  endwhile
end
```

其中,当不满足结束条件时,running 为 TRUE;否则为 FALSE。

5.2.1 主从式(master-slave)并行化方法

当我们考察上述遗传算法时,发现该算法只有在最外层结构(初始化部分)上才是纯串行的,而在内部层次上存在着许多潜在的并发性。例如,当施行适应度评估时我们可以相互独立地评估群体内的每个个体的适应度,从通信量的角度来讲,这意味着在评估进程之间无需通信。如单纯从减少通信量入手,也很自然地首先想到可将适应度评估等局部操作交给从处理器网络(slaves)并行执行,即

```
for i=1 to s par-do
  计算个体 i 的适应度值
endfor
```

其中,s 是群体中个体的个数。

而将选择、交叉等全局操作留给主处理器(master)串行执行,这就是所谓主从式并行化方法。

然而,即使我们拥有和群体大小相同个数的处理器,但是因为无论当哪个处理器运行主算法时都要有同步机制,所以像这样来开发存在于遗传算法中的并发性效率还是不高的。这是由于主进程忙而子进程空闲以及子进程忙而主进程空闲等情况(即负载不均衡)所造成的。当然,如果适应度评估很费时(实际情况通常是这样的),则这么做就能获得令人满意的效率。

在上述算法的并行实现中,选择操作应该对整个群体的适应度有个全局的了解,这部分地决定了通信要求。主处理器必须知道每个个体的适应度值,所以必须支持多到一的通信。对此的一个改进方法是放牧式(farming)。放牧式的思想或结构适用于有一组相互独立的工作可以并发完成的问题,控制进程即运行在根节点上的牧场主(farmer)进程将任务划分为工作包,然后将它们“放牧”到一组相同的工人进程上。用放牧式的思想来实现并行遗传算法是,让牧场主进程保存有整个群体的适应度值,它负责执行遗传操作,而适应度评估工作则交由工人网(workers)完成。接收到任务的第一个空闲的工人把它承担下来,完成它并将结果送回给牧场主。

通常,每个处理器上有一个工人进程,因为从主处理器到每个从处理器间都有直接连接是不可能的,所以必须以一个合适的拓朴(如流水线)结构来连接处理器,并增加选路进程负责给空闲的工人进程送去工作包,给控制进程回送结果。

因为集中存储群体易造成瓶颈,所以任何基于放牧式的并行遗传算法的可扩放性都不好。如果评估的粒度很细,致使通信延迟成了负担,则放牧式就没有好的性能。但如果评估工作所占份量很大,则放牧式就显得比较吸引人了,其理由如下:

- (1) 放牧式很通用且容易实现。
- (2) 如果个体评估需时相同,则放牧式效率很高且负载均衡。
- (3) 可以用放牧式来模拟迁移式和扩散式。

(4) 放牧式将评估交给工人网络来完成,降低了对主处理器的内存要求。

实现时,体系结构可以采用树而不是流水线,控制进程放在根节点上。树比流水线好在随着处理器个数的增多,工人到控制者间的距离成对数增长,而在流水线上则呈线性增长。Transputer 网络可以构成二叉树或三叉树。

但是,这种方式也只有在交叉和评估比选择和传送费时很多的情况下才有效。因为它不是将进化过程并行化的“自然”方式,所以无法得到令人满意的结果。即使只用少量个处理器,放牧式的加速比也很小。

总的来说,主从式比较直观且容易实现,它并没有对标准遗传算法的框架结构作任何改动,所以不会影响其解决具体问题的效果。但是它不可避免地存在有负载不均衡的问题,而且通信量仍然很大,这使得它的效率不高,从而限制了它的实用性。

5.2.2 粗粒度模型

在自然界中,物种的群体由一些个体组成。在处理器个数较少的情况下,我们可以将群体分为若干个子群体,每个子群体包含一些个体,每个子群体分配一个处理器,让它们互相独立地并行执行进化,每经过一定的间隔(即若干进化代)就把它们的最佳个体迁移到相邻的子群体中去。这种粗粒度的并行遗传算法被称作迁移式(migration)或孤岛(islands)模型,可以描述如下:

begin

(1)产生一个初始群体并将它划分成 p 个子群体

(2)for $i=1$ to p par-do

(2.1)初始化

(2.2)评估第一代子群体的适应度

(2.3)while running do

(2.3.1)for $j=1$ to $n(\text{generations})$ do

• 200 •


```

(a)select parents
(b)reproduce
(c)mutate offspring
(d)replace parents      /* 形成新一代子群体 */
(e)evaluate sub-population
endfor
(2.3.2)select emigrants      /* 选择要迁移出去的个体 */
(2.3.3)do step (a) and (b) in parallel
(a)send emigrants      /* 发送要迁出的个体 */
(b)receive immigrants /* 接收迁入的个体 */
endwhile
endfor
end

```

其中, p 是处理器个数,也是子群体的个数。另外,发送要迁出的个体和接收迁入的个体这两个进程是并发执行的,这是为了避免通信过程中的死锁。

5.2.3 细粒度模型

如果并行系统中有足够多的处理器,则我们可以将每个个体分配一个处理器,让它们互相独立地并行执行进化,这样就能获得并行遗传算法的最大可能的并发性。相应地称这种模型为细粒度模型,它可以描述如下:

```

begin
(1)产生一个初始群体并将它分配到  $p=N$  个处理器上
(2)for  $i=1$  to  $N$  par-do
while running do
(2.1)do step (a)and (b)in parallel
(a)receive immigrants      /* 接收迁入个体 */
(b)send self                /* 发送本个体 */
(2.2)evaluate immigrants    /* 对迁入的个体进行适应度评估 */
(2.3)select mate from immigrants /* 从迁入的个体中选择对象 */

```

```

(2.4)reproduce                      /* 交叉 */
(2.5)mutate child                  /* 变异 */
(2.6)evaluate self and child       /* 评估本个体及其后代 */
(2.7)replace self                  /* 用后代取代本个体 */
endwhile
endfor
end

```

其中, N 是群体中个体的个数, 在这里它等于处理器个数 p 。

注意, 这和前面粗粒度模型描述的重要差别在于外层 PAR 循环结构的控制次数不同, 前者是子群体大小, 即处理器个数; 而后者则是群体大小, 即群体中个体的个数。这也是这两种模型的根本差别之所在。

在细粒度模型中, 通常处理器被连接成平面网格(grid), 每个处理器上仅分配一个个体, 选择和交叉只在网格中相邻个体之间进行(根据一个预先定义的邻域结构来判定个体之间是否相邻)。这种细粒度的并行遗传算法被称作扩散式(diffusion)或邻域模型。

上述两种计算模型是目前已经在实际应用中取得了很好效果的并行遗传算法, 它们很自然地分别适用于不同硬件结构的并行计算机: 粗粒度模型适用于 MIMD 机器, 如基于 Transputer 的多处理机系统; 细粒度模型适用于 SIMD 系统, 如阵列式多处理器系统或连接机。另外, 实验证明细粒度模型在基于 Transputer 的多处理机系统上也能高效率地实现。

以下, 我们将分别介绍这两类计算模型的可行性、基本算法以及在并行计算机上的具体实现。

5.3 粗粒度的孤岛模型

事实上, 在自然界一个物种的动物通常不是形成一个大的随机交配的群体(即各个个体间完全随机交配的大群体), 而是分布成若

干个彼此间相互独立的子群体,仅仅在不规律的间隔中才有个体的相互迁移。所以在遗传算法的并行化过程中,为了减少通信量,可以降低全局评估和交叉的频率,不是每代一次,而是迭代若干次以后再全局通信一次。按此,许多研究者不约而同地想到了粗粒度的孤岛模型,他们在不同的实际问题中采用的具体模型在邻域的定义、迁移的频率以及各个子群体的进化参数上有所不同,但算法的基本框架如下:

(1) 随机产生一个初始群体并将它分成 Numpop 个子群体;为子群体定义一个邻域结构。

(2) 并发地对每个子群体 $sP_i (i=1, 2, \dots, \text{Numpop})$ 执行(3)和(4)。

(3) 在所给定的进化代内对 sP_i 执行基因操作,包括选择、交叉、变异等。

(4) 邻域间的个体迁移,行成新的一代子群体。

(5) 若不满足结束条件则转去(2)。

这样,通信量得以大大减少,但对进化效果会造成什么影响呢?下面,我们将分析其可行性。

5.3.1 粗粒度模型的生物学依据

在一个群体中所有可能与某个个体交配的那些个体组成的集合被称为同类群(deme)。如果某个群体中所有成员的同类群等同于该群体自身,则该群体是随机交配的(panmictic),即其中每个个体都有可能与群体中任何一个其它个体交配。前面所述的串行遗传算法就是随机交配的。但自然界并不服从随机交配原则,群体常常是广泛分布的,只有孤立地看各个子群体内部时才是随机交配的。

在古生物学中有一种关于分布式进化过程的“断续平衡”(punctuated equilibria)理论,该理论有两条基本原则:孤立发生的物种形成(allopatric speciation,即当个体被从地理上与他们共同的祖先分开后新物种的迅速进化)和静态平衡即停滞(stasis,即在一

个稳定的环境中达到平衡后,物种的基因组成持续不变)。这两条原则促使研究者们设计出由多个子群体组成的并行遗传算法模型:首先子群体进化几代(称为一个纪元,epoch)后达到稳定(即不再产生更好的个体);然后通过个体迁移去改变子群体的环境后再开始一个局部进化的新纪元。

达尔文进化论有五个基本组成要素,其中的三个是:一个有界的竞争场所、一个不断复制最后必须扩张到竞争场所边界之外的群体和在竞争场所内多种不可避免的因素影响下对空间的竞争。实际的观察报告也证实,那些孤立的环境(例如孤岛)经常会产生这样一些动物物种,它们对自身的环境特质的适应性比在更广阔的地域上产生的相应物种更强。这就导致了下述的假设,即若干个相互竞争的子群体,比起将所有个体都聚在一起的大群体在搜索方面效率更高。

其实,从直观上考虑,自然界的物种进化也正是这种情形。由于受地理条件的限制,在整个全球范围内全局进化是不可能的,物种只能分成若干子群体分别进化,偶尔有个别优秀个体能突破地理限制迁移到邻近的子群体中去。同样,人类在进化之初也是分成了一个部落,现在也受着国籍、地域等的限制。

基于分布式群体的并行遗传算法,最简单的例子是粗粒度的孤岛模型。生物进化史表明,这种孤岛方式不仅没有妨碍正常的进化,反而促成了目前千变万化的适应性很强的物种群体。

5.3.2 粗粒度模型的研究现状

在这一领域,早期的工作之一是 Pettey,Leutze 和 Grefenstette 想用基因搜索的方法求解一个具体问题,但该问题的解的编码表示是一个很长的串,其适应度评估很费时。他们的解决办法是若干个子群体的并行进化(其根据是有些物种包含一些彼此独立的不同的子群体,但它们也能有共同的后代),所用的算法与我们在前面所描述的很相似。^[2]与此同时,Cohon,Hegde,Martin 和 Richards 在一个原始的分布式遗传算法模型上也给出了类似的结果^[3]。

粗粒度模型有几种变化形式,例如可以对不同的子群体赋以不同的控制参数以获得能适应不同环境的进化过程。另外,在个体迁移方面也有一些不同的策略:要迁移的个体可以选择适应度最佳的,也可以只选择适应度不低于子群体内平均值的,或者干脆是随机选择的。在这方面的众多实验得到两个有趣的结论:一是随机选择迁出者所产生的效果至少不比任何其它方法差;二是孤岛模型仅需最小的迁移率就足以获得比随机交配(即串行遗传算法)更好的性能^[5]。对于迁入的个体如何加入子群体中,可以是随机地选择相同个数的个体加以取代,也可以有目的地选择适应度最差或者是和迁入个体一模一样的个体加以取代。后者的效果显然要比前者好。

有些研究者从他们所面临的具体问题出发,对粗粒度的孤岛模型进行了一些改进。如 Kroger, Schwenderling 和 Vornberger 建议为每个子群体保存一个由其邻近子群体的最佳个体组成的个体集,该个体集可以用于交叉、选择等本地进化的遗传操作,但是只有当邻近子群体找到了新的最佳个体并播送过来时才能得以更新^[8]。这样,个体迁移就是不定期发生的了。因为这种改进比较特殊,所以我们将它放在5.6节再作详细介绍。

5.3.3 孤岛模型在 MIMD 机器上的实现

正如前面所提到的,MIMD 机器为孤岛模型提供了最佳的开发环境,同时也能发挥其自身的最大潜力。此外,一些研究者还提出可以对不同的子群体执行不同的遗传算法。以下,我们将概括地介绍分布式模型在这种并行硬件上的最重要的应用,以及近年来报道的这方面的结果。

首先,我们回顾一下孤岛模型的实现。

在孤岛模型中,每个子群体是一个独立的进程,在为这些子群体选择合适的拓扑连接时,可以选择使群体内部距离最大的一种最简单的结构。所选定的结构是将子群体连成环状,迁移可以沿环的一个方向进行,这样在整个群体内流动的基因的最佳量有可能比较低,其

原理来源于对群体遗传的“踏脚石”(stepping stone)模型观察。在这些类似于孤岛型遗传算法的模型中,相邻的子群体间只需要少量的迁移就能产生有效的随机交配群体。

East 和 Macfarlane^[5]选择的迁移方式是每次从子群体中随机选择一部分发送出去,接收到的个体数应该与发出去的个体数相同,而不包含其它迁移方式中的个体复制。虽然比起其它更复杂的迁移方式来,这种不一定有更好的性能,但它确实是个简洁的方式,能把迁移和选择完全结合到一起。

两次迁移之间的迭代次数和迁移中交换的个体所占比例是孤岛模型里新增加的参数,这在标准遗传算法中是没有的。

孤岛模型的进程可用如下的 OCCAM 伪代码进行描述,其中“...”是 OCCAM 中的折页符号,它将一段程序打包,然后用“...”后的一个标题来代替它,相当于在程序清单中将这段程序折叠起来了。用功能键可以进入折页进行翻阅或修改,这是 TDS(Transputer Development System)所提供的一个很有用的编辑功能,它能使 OCCAM 程序简洁、易读。

```
... host, interface process procedure
... subpopn process procedure
... monitor, subpopn process procedure
[N]CHAN OF MIGRATION M; /* 迁移通信链 */
CHAN OF CONTROL to, subpopns, from, subpopns; /* 控制通信链 */
PAR
    host, interface(to, subpopns, from, subpopns)
    monitor, subpopn(M[0], M[N-1], to, subpopns, from, subpopns)
    PAR i = 0 FOR N-1
        sub-popn(M[i], M[i+1])
```

其中,宿主界面进程是用来提供和宿主机的 I/O 通信,与它连接子群体称为监控程序段(monitor, subpopn)子群体,它负责将环中得到的结果回送给界面进程。

由 N 个子群体构成的环状配置如图 5.1。

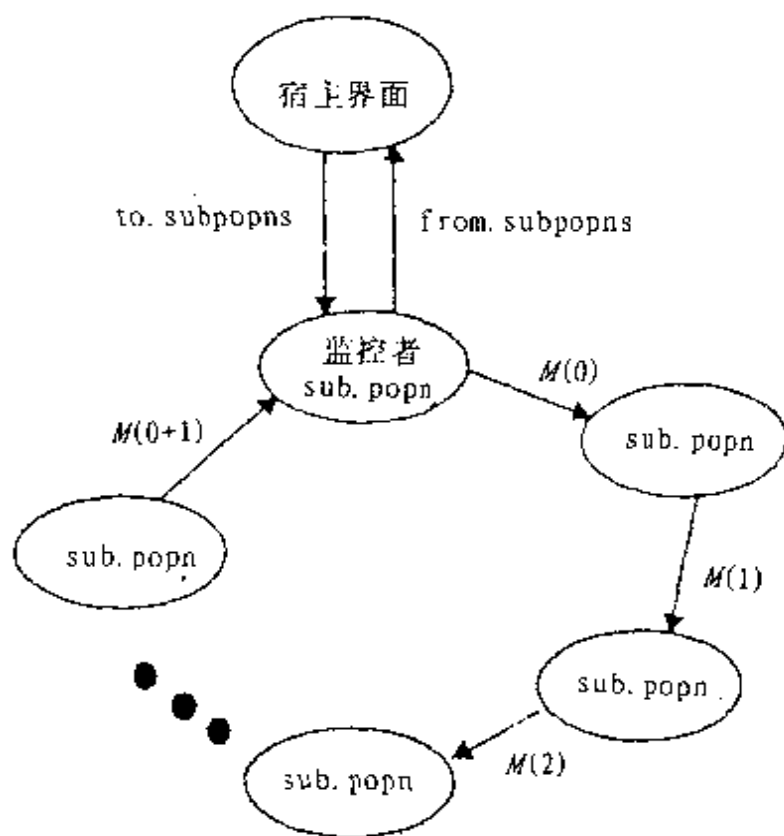


图 5.1 N 个子群体的环状迁移 GA

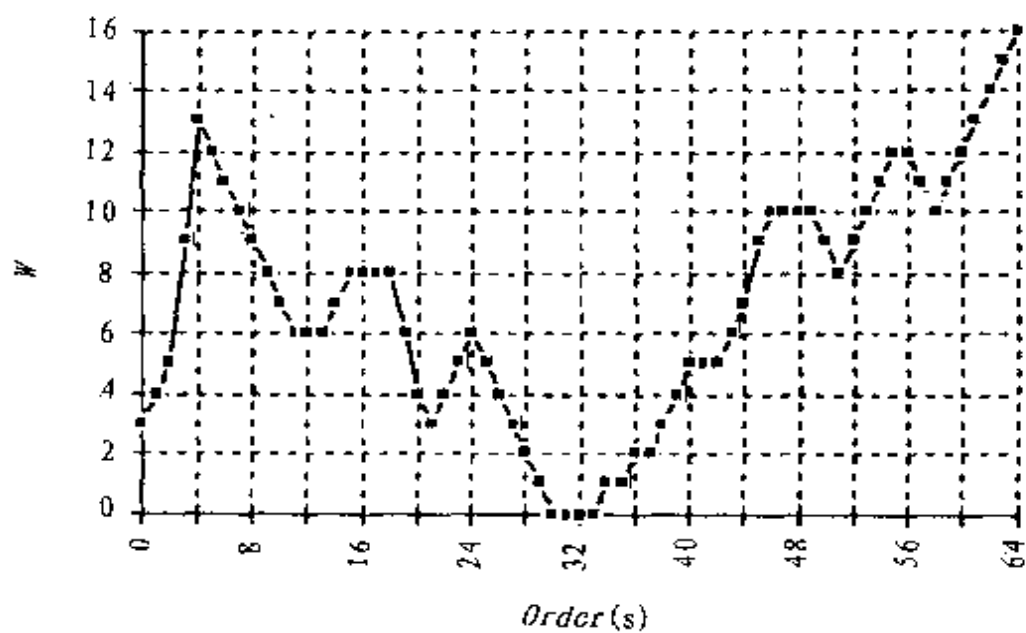


图 5.2 Walsh 函数

Tanese 等^[4]首先在一个超立方结构上实现了孤岛模型, n 维的超立方是由 $N=2^n$ 个处理器构成的 n 维二进制立方体。每个处理器作为立方体的节点拥有自己的 CPU 和局部内存,处理器间的通信方式是消息传递。每个处理器都和其它 n 个作为其邻域的处理器直接相连,这使得立方体中任何两个处理器间的距离最大为 n 。他们采用的是 NCUBE 公司生产的 64 个(即六维超立方)处理器的通用 NCUBE 系统,其中每个处理器都是定做的 32 位类 VAX 芯片,带有 128K 局部内存,其中四分之一用于存放节点操作系统和消息缓存。

每个处理器负责一个子群体,并间断性地与其邻域交换最佳个体,所用的测试函数是如图 5.2 所示的 Walsh 函数, $W(\text{order}(s))$, 其中 s 是 64 位 0/1 串, $\text{order}(s)$ 是 s 串中的 1 的个数。

当保持群体大小不变(400 个个体),增加超立方的维数(即增加处理器个数)时我们发现,每个子群体迭代次数之和大致等同于一个串行实现要获得最佳结果所需的迭代次数,但这里没有考虑并行算法挑选好个体去取代差个体以及子群体间交换个体所造成的额外时间开销(实验证明它们分别占总的执行时间的 1% 和 2%)而只考虑了迭代次数。在恒定的迭代次数之内获得最大值,同时每代的计算时间又等同,但又是分布在若干个处理器上,这意味着达到了线性加速比。如图 5.3 所示,其中仅当 $N=64$ (即 $n=6$) 时加速比小于线性,这是因为此时每个处理器上仅有 6 个个体,通信负担过重的缘故。

他们还针对参数设置(即交叉、变异的概率)做了些实验,但与 de Jong 所推荐的参数设置不相同,而且不同子群体的参数也设置得不相同。所得结果与最佳设置下获得的结果差不多,这说明在参数设置方面分布式算法比串行算法有更好的鲁棒性,这是一个长处。

Pettey 和 Leutze^[6]也在一 8 个节点的超立方机器上实现了并行遗传算法,他们通过实验验证了这样一个经验主义的假设:即在每个处理器分配一个子群体的孤岛模型的实现中仍然保持着串行遗传算法中由模式定理保证的效果。在前面所讨论的理论工作的指导下,实验证明了这种模型的效果。

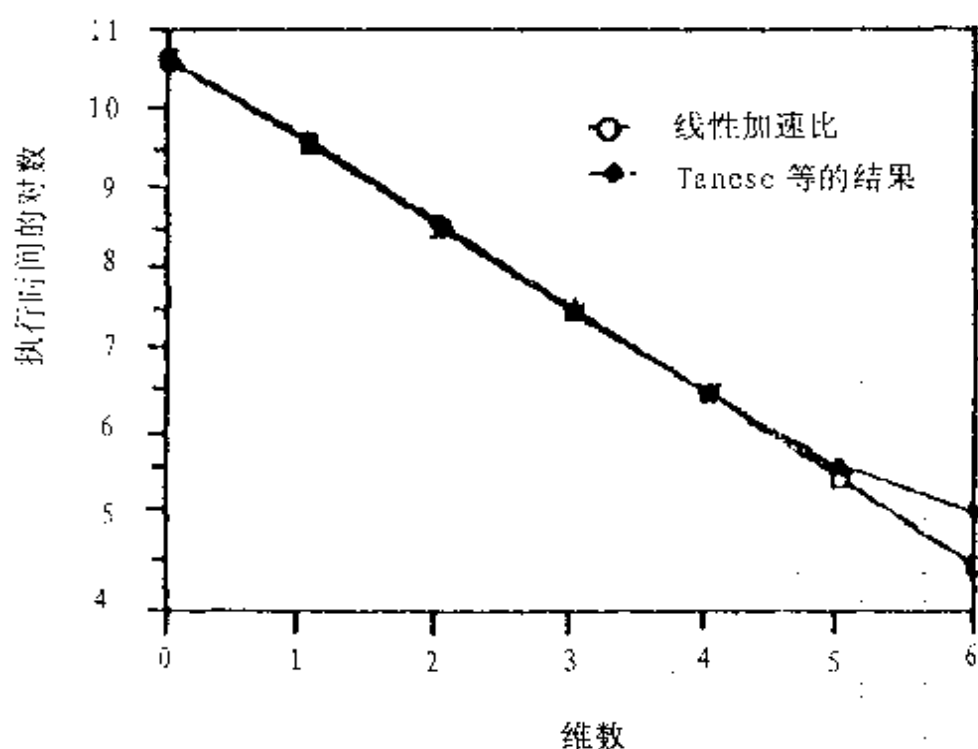


图 5.3 加速比曲线(处理器数和执行时间均取对数)

对遗传算法的并行实现效果的验证工作是由许多人完成的。例如 Brown, Huntley 和 Spillane^[7] 将他们的 SAGA 算法在 32 个节点的超立方上的实现与最快下降的启发式方法作了比较, 选用的实例是流量分配问题。SAGA 是遗传算法(用于识别搜索空间的有希望区域)和模拟退火(用于围绕着由遗传算法提出的种子的精细调节)的结合。当在超立方计算机上实现时, SAGA 在并行化的加速比和所能达到的解的质量方面都是很有有效的, 它的加速比也接近线性。在用于较难和较容易的问题时, SAGA 都获得了比启发式更好的效果。

Kroger, Schwenderling 和 Vomberger^[8] 用 MIMD 实现的孤岛模型遗传算法处理二维组装问题, 即将一些小矩形放到一边无界的大矩形中, 要求使水平方向占据的长度最小。在他们的子群体中不仅有本地个体(称为内部个体), 而且也有外部个体(由邻近子群体中挑出来的最佳个体)。他们的硬件结构是 32 个 Transputer 连成的环状结构, 其中加有几条弦用于减小直径。研究目的是要搞清楚对于搜索

而言是小邻域(如2个子群体)好还是大邻域(如10个子群体)好,以及交叉时是从其它子群体中随机挑选一个对象为好还是选其中最佳者为好。经过实验所得到的结论是:小邻域、大邻域效果几乎相同;随机选择策略更有效。

Cohoon, Martin 和 Richards^[9,10]在超立方结构上处理 k 划分问题,即将有向图分成 k 个部分,要求进入和离开每个部分的链最少。他们采用带有断续平衡的遗传算法系统来将200个结点的图划分成15个部分。一个标准的遗传算法用来和运行在16个节点的超立方上的并行遗传算法作比较,使前者的迭代次数是后者中每个子群体的迭代次数的16倍。结果表明,在每次测试运行中,并行遗传算法都优于串行遗传算法,它总能找到最佳解。另外,它还充分利用了其并行硬件系统,因为它的运行时间只有串行遗传算法的16分之一。

Muhlenbein, Schomisch 和 Born^[11]将并行遗传算法用作函数优化器,他们的 PGA 系统实现的孤岛模型带有基于序(rank-based)的选择机制,并增加了一个从遗传产生的解出发的贪心局部搜索算子。他们的硬件结构是一个如图5.4所示的搭成梯子状的多处理器 MIMD 机器“megaframe hypercluster“,这种体系结构在这一领域中应用很广泛。

他们从达到最优解所需的时间和函数求值的次数这两方面来衡量系统的性能。比起标准遗传算法来,PGA 在绝大多数函数上都有更成功的结果。

他们对于处理器个数增加时的加速比的研究也极感兴趣。实验显示,在较容易的问题上所得加速比很小,这可以解释为问题太简单以至于不能充分利用并行硬件的潜在能力。事实上,增加处理器个数对困难些的问题在效率上能获得明显的增益。特别值得注意的是,在从4片增加到8片时曾出现了超线性的加速比。更多的处理器(如64个)可使他们能处理很大的问题实例,甚至是以前从未碰到过的。几个松散连接的子群体构成的分布式遗传算法的搜索能力再次被证明是解决优化问题的好方法。

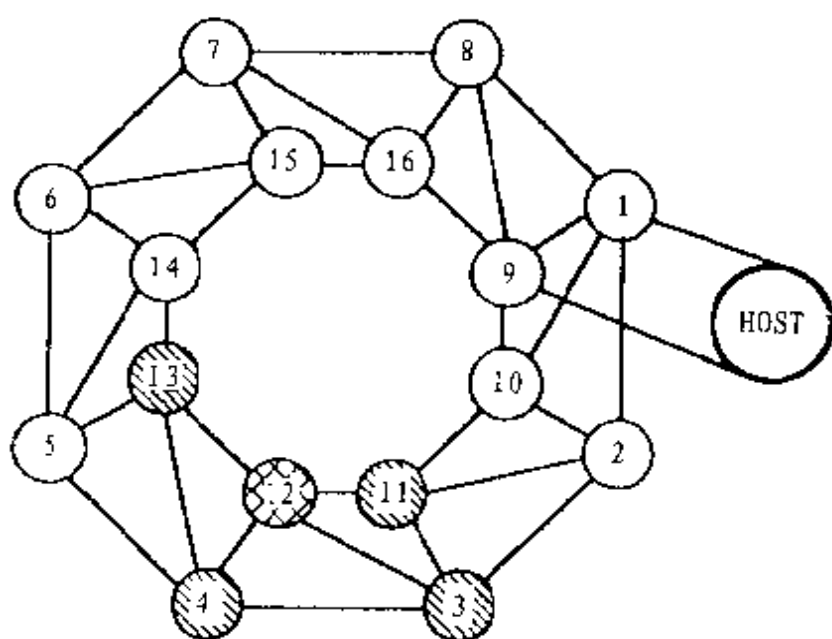


图 5.4 梯子状的多处理器系统

5.3.4 扩展的分布式遗传算法

虽然比起主从式来粗粒度模型有较高的加速比,但是最佳个体的多次迁移仍然造成了一定的通信开销,这使得其加速比达不到线性。而且因为选择的局部性,获得的解很有可能不是全局最优的。

Prahladah 和 Hansdah^[12]提出的扩展分布式遗传算法(EDGA)是粗粒度模型的推广,它采用分布式子群体的概念去获取好的加速比,同时又采用全局搜索策略的概念去获取较好的最优解。根处理器执行对整个群体的全局选择,而处理单元(PE)执行对相应子群体的局部搜索。经过一个预定的间隔,根处理器和 PE 上的经过进化的群体相互交换,此周期重复循环一定次数或直到满足终止条件为止。子群体的局部适应和整个群体的全局适应结合起来提供了一种在最佳搜索空间的快速搜索和找到更好的最佳解的能力。EDGA 简单、高效,在现有的并行硬件系统上可以很容易地实现。这种算法可以获得很高的加速比,而且它还能使所有处理器达到负载平衡,从而保证并行系统能有好的性能。

1. EDGA 的基本术语

在描述 EDGA 算法以前,我们先对它所用的一些术语说明如下:

总群体:放在根处理器上,其大小为 PopSz。

子群体:总群体被分成大小为 $\text{SubPopSz} = \text{PopSz} / \text{PE.num}$ 的子群体,它们被分配到各处理单元(PE)上,其中 PE.num 表示 PE 的数目。

归并子群体:进化后的子群体被从 PE 传送到根处理器上,归并形成新的总群体。

全局选择:从总群体中选择个体。

局部选择:从子群体中选择个体。

终止条件:已达到已知的最佳适应度值,或固定的迭代次数 G 等。

2. EDGA 的算法描述

Randomly generate an initial population P_0 of size PopSz

Divide the Population into PE.num subpopulations of size SubPopSz

Copy one subpopulation to each PE

Send the parameter values to all the PEs

for $i=1$ to K do

 On root run SGA on P_0 until all PEs interrupt

 for all PEs par-do

 for generation=1 to G do

 Run sequential GA on subpopulation

 endfor

 Send interrupt to the root processor

 Send evolved subpopulation to root

 endfor

 Make population P_i with evolved subpopulations

 Make subpopulations with evolved P_i

 Send one subpopulation to each PE

$P_0 \leftarrow P_1$

endfor

初始, EDGA 在根处理器上随机地产生一个初始总群体 P_0 , P_0 的一个备份被划分成若干个子群体 $sP_{01}, sP_{02}, sP_{03}, \dots$, 并分配到各处理单元(PE)上。根处理器对 P_0 执行串行遗传算法(SGA)得到 P'_0 , PE 对各自的子群体 sP_{0i} 也执行串行遗传算法, 经过一定代数(G)的迭代后得到 sP'_{0i} , 向根处理器发一个中断。根处理器将所有的子群体 sP'_{0i} 归并成一个新的总群体 P_1 , 而将 P'_0 分成子群体 sP_{1i} 分配给各 PE。接着, 根处理器和诸 PE 分别对 P_1 和 sP_{1i} 执行串行遗传算法, 将这个过程重复预定次数(K)或直到满足结束条件。

3. EDGA 的性能分析

在所有 PE 上, 选择都是局部的, 即各 PE 仅从本地子群体中选择个体, 这使得 EDGA 不会过早地丢失那些暂时显得不利(称为次最佳)的基因信息。因为这种次最佳的信息可以保留一段时间, 所以其中真正好的基因元素(如果存在的话)就有机会在将来的进化过程中为全局的最佳解贡献出它们的好的基因。同时, 根处理器依据全局选择策略(即从总群体中选择个体)对总群体执行串行遗传算法, 这有助于防止算法陷入局部最优。

局部适应的子群体和全局适应的总群体的结合为 EDGA 赋予了一种能力, 即既能收敛到全局最优又在 MIMD 机器上有很好的加速比。在 EDGA 中, 根处理器一直是忙碌的, 所以能将并行计算机充分利用起来。

EDGA 中强调的是全局选择、提高加速比和充分利用根处理器, 它的优点有:

(1) 在根处理器和处理单元(PE)之间达到负载平衡, 提高了处理器的利用率。

(2) 同时具备粗粒度模型的长处和主从式的全局选择的特点。

Prahlada 和 Hansdah 在基于 Transputer 的并行 MIMD 机器上对主从式、分布式和扩展的分布式遗传算法进行了对比实验。他们选

择的体系结构是超立方,分别由1,2,4,8,16片 Transputer 组成了0,1,2,3,4维超立方体。对通信时间的测量他们用了两种方式,一种是不包含发送和接收者之间因需要同步而导致的等待时间;另一种是把这种等待时间也计算在内。从他们的实验数据来看,在 EDGA 和 DGA 中计算时间都明显地大大高于通信时间。同时当处理器个数增加时,DGA 中的计算时间在增加,而 EDGA 中的计算时间则在下降。实验数据还表明,EDGA 的加速比总是比主从式的加速比高。如果考虑到等待时间的话,则当处理器个数大于5时 EDGA 的加速比就高于 DGA 了;如果排除等待时间,则 EDGA 的加速比一直高于 DGA。

5.4 细粒度的邻域模型

大规模并行硬件系统最适用于处理那些依据简单局部规则而工作的分布式模型,而细胞自动机模型代表着分布式计算的很通用的框架。将遗传算法与细胞自动机相结合已被证明对自动机理论是很有用的,因为这种结合为学习局部更新规则并且使它们适应实验数据提供了一条可行的途径;另一方面,这也为遗传算法提供了一个高效的并行计算框架。

这样产生的分布式计算模型包含有空间性的概念,遗传算法中群体里的个体被分配到给定空间环境(通常是排列成环形阵列的二维网格状以防止边界效应)中的特定位置。所以网格上的邻域关系就限定了个体间空间上的关系。遗传操作被看作随机的局部更新规则,这样该模型就是完全分布的而无需任何全局控制结构。

对每个细胞而言,选择仅在赋给该细胞及其邻域的个体上进行;交叉也仅交配邻近的个体;而变异是标准的。另外还增加了其它操作。细粒度模型上的算法的基本框架如下:

(1) 随机产生一个初始群体,并将个体一对一地分布到网格中的细胞上,计算每个个体的适应度值。

(2) 并发地对每个细胞 c 执行(3),(4),(5)和(6)。

(3) 从 c 及其邻域中选择一个个体占据 c 。

(4) 从 c 的邻域中选择一个个体,根据概率 P_c 使其与 c 上的个体交叉,再将子个体之一赋给 c 。

(5) 对 c 上的个体根据概率 P_m 进行变异。

(6) 计算 c 上的新个体的适应度。

(7) 若没结束,转去(2)。

因为遗传操作都成了局部操作,所以通信量大大降低。下面,我们将首先分析这种细粒度的邻域模型的可行性。

5.4.1 细粒度模型的理论基础

细粒度模型上的遗传算法也叫作扩散式(diffusion)并行遗传算法,因为在此模型上同生类(细胞的邻域就是该细胞上的个体的同生类)的重叠可以使适应度高于平均值的模式(scheme)慢慢地扩散到整个群体。

已如上述,细粒度模型可以看作是一种细胞状的自动机网络,群体中的每个个体相当于一个自动机,其状态代表了一种基因型(genotype),局部遗传操作控制着自动机状态的更新。实验表明其中子代对父代个体的取代(replacement)对并行遗传算法的效率和性能都有深刻的影响。可选择的取代方式有:

(1) 直接取代。

(2) 仅当子代个体较优时才用它取代父代个体。

(3) 根据父代和子代个体的适应度差别而或然地取代(模拟退火)。

如果选择(2)而不是(1)则性能会有改善,这样做也无需任何额外开销,因为无论选择何种方式,适应度值都是必须计算的。

这种细粒度模型令人想起人工生命领域的有关 animat 的早期工作^[13]。Sannier 和 Goodman^[14]等人早已尝试将一代基因型(即个体)分配到二维世界上,用遗传算法使之进化以得到适应环境的个

体,适应度由特定的适应度函数值表示。和所有类似的 animat 工作一样,他们让群体中的个体在网格上移动,适应度是它们在游历途中吃到的食物个数的函数,这些食物都被转化成了内部能量。群体大小在动态地改变,因为个体的能量越多,它们所产生的后代也越多,当个体的内部能量小于给定的阈值时它就会死亡。如果我们给每个网格细胞分配一个个体,从而固定了群体的大小并禁止游历,再将个体当作给定问题的一个解代码(即基因型),将适应度度量定义为这种基因型的优劣程度,这就形成了遗传算法的细粒度模型。

5.4.2 细粒度模型的研究现状

和孤岛模型一样,有若干研究者分别独立地找到了这种细粒度算法模型。

Manderick 和 Spiessens 首先比较了并行遗传算法和标准算法的性能^[15],结论是细粒度模型能提供对搜索空间的更彻底的探测(exploration),因为它的局部选择机制减轻了选择压力(这好比几个人搜索一大片地域,如果他们分工包干,一人负责一小块地方,则会比大家合伙干的效率要高)。这样,简单问题的求解效率可能不高,因为新增的探测仅代表新增的计算负担而并没有好处;但难度较大的问题将从这种彻底搜索中获益。另外,他们还首次给出了关于算法对参数设置的敏感性的分析结果。最有趣的发现之一是:在并行机上的遗传算法没有一个最佳的群体大小;机器处理能力越强,群体可以越大而且结果越好(与串行算法到目前为止所能给出的最佳结果相比)。其原因是既然可以给每个个体分配一个处理器,那么在并行计算机上不同群体的运行次数是与群体大小无关的。第二个有趣发现与邻域尺寸有关:最佳尺寸因具体问题而异,邻域越大,并行遗传算法的性能就越接近串行算法,但小邻域就足够保证能获得良好的性能了。

Muhlenbein 对邻域模型提出了一些显著的修改:每个个体都作局部的爬山(hill climbing);选择基于一个列队(ranking)的过程(即

将群体的个体按序排列);交叉成为基于投票过程的一个重组操作^[16,17]。

将局部的爬山看作是一种新的基因操作,要放在所有其它操作之前执行。这样,选择、交叉、变异等都只是在局部最优空间进行,得到的子代个体普遍说来将不是局部最优,但若被用作贪心的启发式搜索的起点,则将可能导致产生越来越好的局部最优解。这种方式通常用来求解组合优化问题,如流量分配问题。

交叉也被改造成重组操作,即若干个邻近个体合作产生一个后代。如果父代个体的重复位(即基因)个数超过能保证相应置换的可能性的阈值,那么后代个体在置换的每一位上重复父代基因的可能性就更大。

Gorges-Schlenter 和 Laszeweski 都用这种系统求解了一些组合优化问题(如 TSP、图的划分等)^[18~20],其中用到了更为标准一些的交叉操作。所得的结果与前文提到的一致:对困难的问题,细粒度遗传算法比标准算法的求解效果好,也较为不易陷入局部最优;考虑到参数的设置,细粒度遗传算法的鲁棒性也较好。

5.4.3 MIMD 上的细粒度模型的实现

对于扩散的 GA,很自然的是一个个体使用一个进程。当考虑在 Transputer 网络上如何分配进程时,East 和 Macfarlane^[5]再次选择了简洁的方式。他们不是给每一个处理器分配一个进程,而是如图 5.5 所示,将诸个体分成几个片(slice),每个处理器上分配一个片,处理器连成环状。

为了提高效率,他们将一个片上的一些个体进程组合成单一进程,串行地更新一个个个体。这就减少了每个处理器上的内部通信,同时提高了应用的粒度,这是为了能最大程度地利用硬件系统的处理能力。在这样的安排下,处理器间的通信是“局部”的,因为只有位于片边缘上的个体才需和其它处理器上的个体进行通信。

Muhlenbein 和他的小组在此课题上作了广泛的研究工作^[16]。他

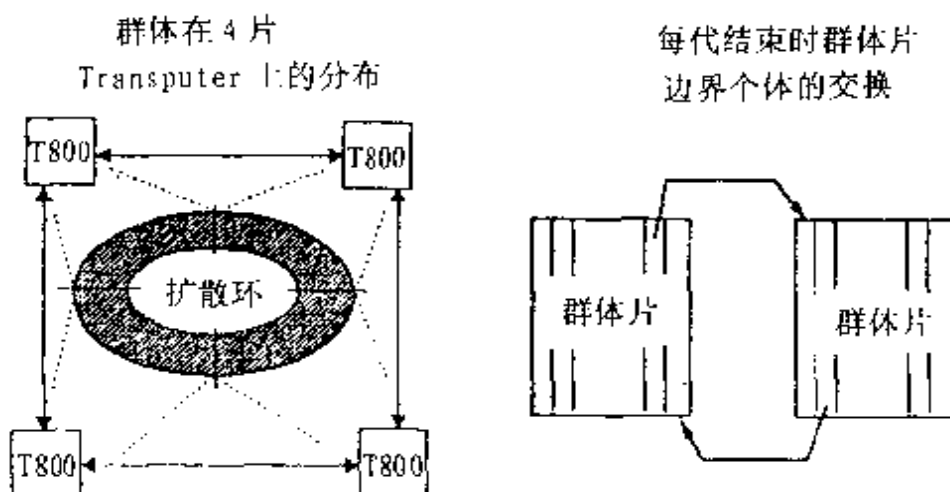


图 5.5 扩散 GA 的环状结构

们将个体分布在由两个相连的环(称作梯状)组成的邻域结构上(见图5.4),这种结构被直接映射到实验所用的硬件上:64个 Transputer,带有分布式内存,按特殊的应用要求形成互连的配置。这个结构被用于多个组合优化问题的测试。例如,Muhlenbein^[17]发表了他在流量分配问题上所获得的结果。他选择的是目前为止规模最大的36维的流量分配问题,结果成功地找到了最佳解。他没有做与串行遗传算法的对比实验,但系统的加速比可由表5.1中的数据看出。

表 5.1 36维流量分配问题的最佳计算时间(秒)

| 处理器数 | 64 | 32 | 16 |
|------|------|------|------|
| 最快 | 279 | 502 | 580 |
| 最慢 | 1151 | 1711 | 3256 |
| 平均 | 579 | 1794 | 1465 |

Gorges-Schleuter^[18]用同样的体系结构求解了不同实例的 TSP 问题,其中规模最大的为532个城市。研究的目的是对算法在不同的参数设置时的鲁棒性,以及与适用于 TSP 的简单启发方式作个性能比较。结果表明,只要有合适的硬件环境,并行遗传算法在解的质量

和所需时间上都不会比其它启发式方法差。在参数设置方面,高的变异率会使收敛变慢,而交叉中交叉点的位置对收敛没什么影响,只要它保持在一定的间隔中,如 $[N/3, \dots, N/2]$,其中 N 为串的长度。另外,较大的问题规模更有可能给出好结果,而且基于适应度值的局部概率选择比自适应(adaptive)和挑优者的选择策略都要好。

另有一些研究者针对同样的实例研究了在不同空间结构上的表现。如图5.4所示的梯状结构被拿来和一个环面结构作比较,该环面结构有较高的连接度,所以个体间的最大距离较小。对比实验表明,较高的连接度可使最差的解很快地被较好的解取代,从而较早结束搜索过程。另一个与标准遗传算法(具有完全随机交配的群体结构)的对比实验表明,在多处理器环境上一个结构化的群体能获得线性加速比,而串行遗传算法在单处理器环境中有效但在多处理器环境中因有同步要求而遇到通信延迟。

Muhlenbein 和 Laszewski^[20]也用这种系统处理图的 k 划分问题,仅在目标函数的计算方面与 Cohoon, Martin 和 Richards 不同。在此问题中,两个适应度值很高的解经组合会产生适应度值更高的解,而且事实上局部的贪心搜索和遗传算法的结合在此会使系统获得很好的结果。同时,他们的实验再次证明小邻域在大群体中会产生最佳结果。

Talbi 和 Bessiere^[21]在他们的基于 Transputer 的环境上实现的细粒度模型中也处理了图的划分问题,群体被映射到一个带环网格状的处理器连接图上,每个处理器上一个个体,机器结构如下图5.6

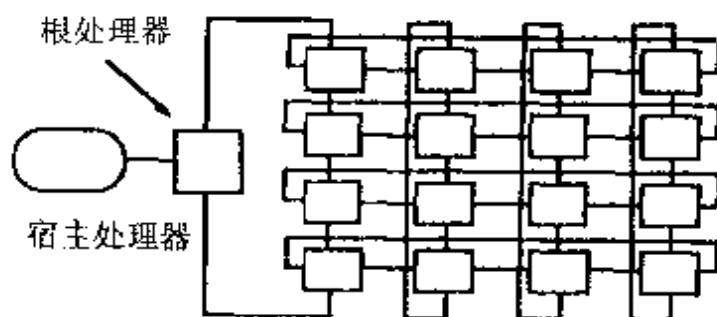


图 5.6 多个处理器的环面结构

所示,采用的遗传算法是标准的没有的局部搜索的细粒度模型。

该体系结构的加速比是以单处理器上的执行时间与多处理器上的执行时间之比的方式来测量的。当处理器个数增加时,在其中某些阶段加速比很明显是超线性的。另外,当应用在相同问题上时,该系统无论在解的质量上还是在达到最佳解所需的时间上其性能都优于爬山法和模拟退火算法。

5.4.4 SIMD 上的细粒度模型的实现

在细粒度模型遗传算法中,很自然地一个个体就有一个进程,为之选择的拓扑结构可以是带环绕的二维网孔结构,其中每个个体都与位于其上下左右的4个近邻相连接。通过改变网孔的各维大小可以修改拓扑直径,从而可以改变流遍群体的基因。两维大小相同时可得到最小的直径,当其中一维增大而另一维减小时,拓扑直径也随之变大直至最终变成环状拓扑。

遗传算法的 SIMD 实现就是并行化和向量化的遗传算法。Manderick 和 Spiessens^[15]在这方面作了很多工作,而且取得了重要成果。他们将该算法与标准串行遗传算法作了性能比较,采用的是如下所述的由 De Jong 所提出的性能标准:

(1) 在线性能标准,即平均的函数评估均值:

$$\text{online}(T) = \sum_{t=0}^T \frac{\text{mean}(t)}{T}$$

(2) 离线性能标准,即最佳的函数评估均值

$$\text{offline}(T) = \sum_{t=0}^T \frac{\text{best}(t)}{T}$$

其中, $\text{mean}(t)$ 和 $\text{best}(t)$ 分别为在第 t 代时群体的平均适应度和群体中最好个体的适应度。

测试是在搜索 De Jong 的最小值过程中进行的,所得的结果与后来 Muhlenbein 所发表的在性质上是一致的,即简单函数太简单了,不能充分利用并行硬件环境的潜力,所以这时并行遗传算法的性

能比串行的略差;而在难度较高的函数上,并行遗传算法明显优于串行的。他们选择了一个遗传算法中的困难函数,即一个自变量为长为64位的二进制串 s 的类 Walsh 函数,系统的目标是要找函数 $W(s) = F(\text{Order}(s))$ 的最大值,其中 $\text{Order}(s)$ 是串 s 中1的个数。在求解这个问题时,细粒度模型的结果比串行的要好。

所以,从搜索效率的角度来看,细粒度模型的遗传算法似乎比串行遗传算法更合适。因为局部搜索使细粒度模型比标准串行的选择操作有更低的选择压力,所以应用在多峰函数上所得搜索效果更好。这一解释也适用于小邻域,因为它使计算时间缩短。不难理解,邻域越大,细粒度模型越近似串行遗传算法。

后来,他们又在分布式阵列处理器(DAP)上实现他们的细粒度模型时获得了一系列结果。他们的 DAP 是一个 32×32 个处理器排列成二维环状网孔结构的大规模并行 SIMD 计算机,每个处理器有可直接存取的局部内存。在这种机器或与此类似的其它阵列机上能期待获得的加速比是以下三个参数的函数:群体大小 n 、邻域大小 s 和基因型的长度 l 。实验结果如表5.2所示,其中选择操作有两个复杂度,这是因为可以有若干种不同的选择策略(如按比例选择、局部轮换选择等)。

表 5.2 时间复杂度比较

| | 串行 GA | 细粒度 GA |
|----|-------------------------------|-------------------------------|
| 选择 | $O(nl)$ $O(n \log n + nl)$ | $O(s+l)$ $O(s \log s + l)$ |
| 交叉 | $O(nl)$ | $O(s+l)$ |
| 变异 | $O(nl)$ | $O(l)$ |
| 总计 | $O(nl)$ $O(n \log n + nl)$ | $O(s+l)$ $O(s \log s + l)$ |

事实上,当基因型长度 l 增加时,群体大小 n 也必须增大,这会

影响到分布式算法的加速比。因为群体大小的增大在标准遗传算法中会使执行时间有多项式级的增加;在拥有邻域大小不随群体大小而改变的细粒度遗传算法中则会使执行时间呈线性增加。实验数据也证实了这种分析,例如图5.7显示出执行时间是基因型长度和邻域大小的线性函数。

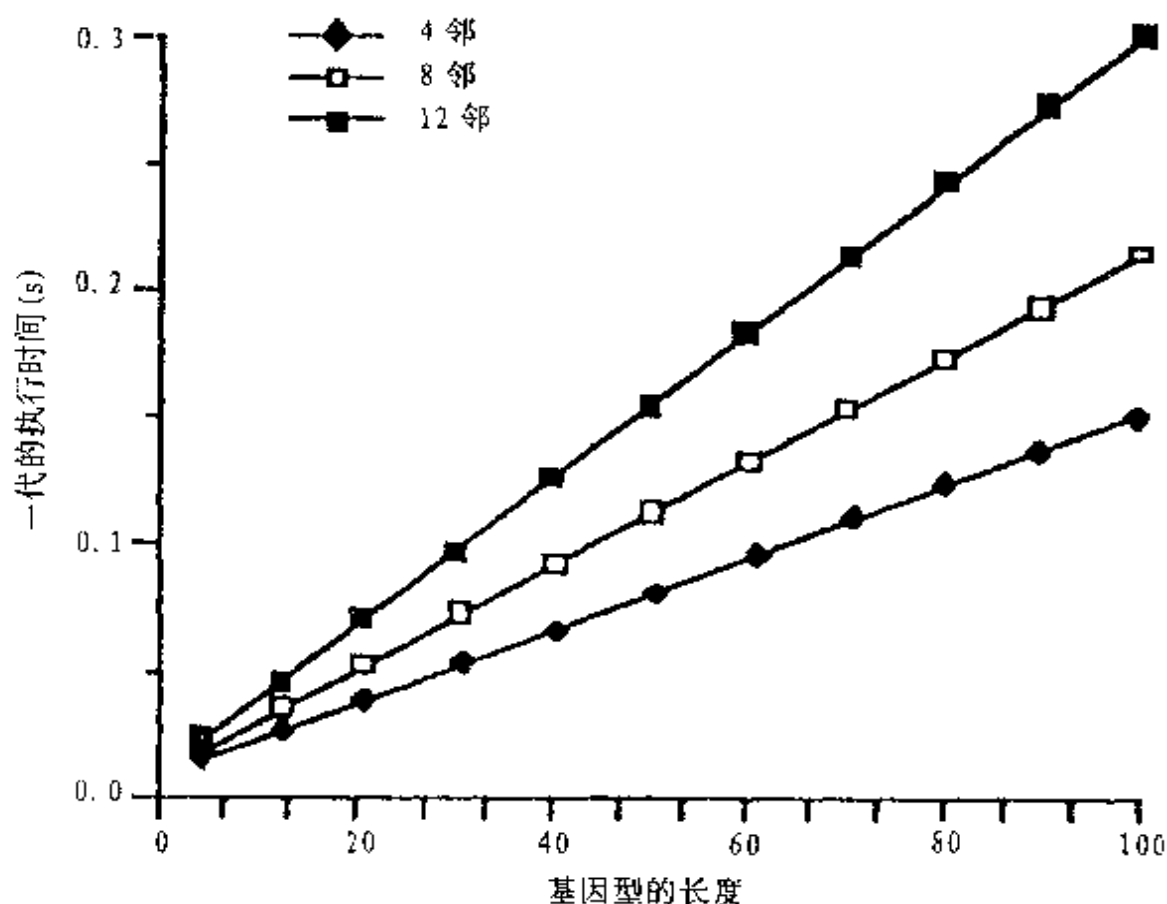


图 5.7 1024个个体进化一代的处理时间

5.5 粗粒度模型与细粒度模型的性能比较

许多研究者都认为粗粒度模型和细粒度模型在效果上并没有重大差异。唯一的差别仅在于,细粒度模型更有利于局部子群体保护自身特质并能防止与其它子群体的基因材料相混杂。局部群体信息的这种小生境性质很有用,因为我们可以看到在一定程度上隔离的子

群体比起随机交配的大群体来更具有基因信息的多样性,而这种基因信息多样性的存在有助于防止算法的提前收敛。

许多文献经过分析,认为粗粒度模型是最好的。如 Tanese^[4] 用一个在超立方并行体系结构上实现的粗粒度模型求解函数优化问题,在总的群体大小保持不变的情况下,处理器规模一直扩大到64个时都能获得几乎是线性的加速比。此外,他们还对不同的处理器采用不同的变异和交叉概率,实验证明确实能改善收敛速度。Petty^[2] 等人也用超立方上的粗粒度模型求解函数优化问题,不同的是他们保持每个处理器上的子群体大小不变,结论是增加处理器个数确实能加快收敛速度,但对解的质量并无改善。

Muhlenbein 的研究小组的工作推动了细粒度模型的研究进展。他们设计了一种选择机制,可以让每个个体从它的二维邻域中选择一个对象。他们采用的网络拓扑是包含了64个处理器的环行梯子状结构。这种结构具有相当大的直径,它为群体对基因的变化性的保持提供了有力的支持。

局部子群体对某些基因的统治权的保持,已被公认为是防止出现过早收敛的关键,这似乎与粗粒度模型的那种将最佳个体同时一齐发向邻近的几个子群体的策略相矛盾,但这种策略能促使局部群体去汇聚形成全局的最佳个体。

此外,在细粒度模型中任何个体的同生类都仅仅是它的邻域(扩散式和随机交配在同生类规模上是完全对立的两个极端),所以算法与它的实现以一种相当有趣的方式联系在一起,即这里存在的隔离度与同生类规模之间的矛盾。

在直接配置(即给每个处理器映射一个个体进程,每条通信链映射一对通信通道)情况下,处理器网络的互连结构确定了每个个体的邻域。如果算法能形成一个规则的进程图并使之等同于处理器的互连图,则配置就是一件相当简单的事了。个体进程的邻域和处理器图一样,而邻域则构成同生类。对任何给定大小的群体即网络,度(degree)比较大的图能提供较大规模的同生类,然而大的度通常意

味着小的直径。实验表明,直径的改变会影响并行遗传算法的效率,特别地,并行遗传算法在解空间搜索最佳复合适应度的能力似乎随着直径的扩大而有所提高,在这一矛盾面前算法该怎么办?回答是孤立型(isolation)。事实上,生物学家认为,在一个群体内的隔离对进化有深刻的好处。随着群体内部隔离度的提高,进化基本上会变得更快速并且较为不容易受阻。从算法的角度来讲,是因为这时搜索更容易摆脱比较不利的局部最小,从而可能避免过早的不成熟的收敛。

5.6 实现并行遗传算法的一个例子

现在,我们来介绍一下 Kroger, Schwenderling 和 Vornberger^[22]在基于 Transputer 的 MIMD 系统上实现的遗传算法的多种并行模型。因为他们的作法与一般的并行遗传算法有所不同,颇有特色,所以专辟一小节讨论之。

5.6.1 迁入式算法

迁入式(immigration)是孤岛模型的一种,其中个体交换是不定期地发生的,作用是更新邻域群体的遗传物质。在每个处理器上除了有一个大小固定为 n 的内部群体 Π_{int} 外,还有一个包含有其邻近处理器找出来的最佳个体的备份的集合 Π_{ext} , 这为当地处理器提供了一种对非本地个体的有限制的存取。邻域在此不仅指拓扑结构上的邻近,而且包括所有和本处理器交换个体的其它处理器。通常, Π_{ext} 的基(cardinality)应该远远小于处理器个数, Π_{ext} 的基以及每个邻域的组成都参数化了。在这里,有限的邻域概念也来自自然界,它模拟了大多数动物只从一个小的周围环境中选择对象的情况。

每个处理器都保持有从本地个体中产生的适应度值最大的个体,当该个体要被新的最佳者 Ω 取代时,就将它发送给那些其邻域中包含该处理器的所有处理器。一旦接收到自近邻 i 发来的改进了的最佳个体 Φ , 每个处理器都修改其 Π_{ext} 集, 用 Φ 取代原先的处理器

i 的最佳个体。所以在每个处理器上的迁入遗传算法可描述如下:

```

local.best =  $\emptyset$ 
 $\epsilon(\text{local.best}) = 0$ 
generation = 0
initialize local population  $\Pi = \Pi_{\text{int}} \cup \Pi_{\text{ext}} = \Phi_1, \Phi_2, \dots, \Phi_n \cup \emptyset$ 
while generation < max.generations do
    (1) generation = generation + 1
    (2) select individuals  $\Phi_i$  and  $\Phi_m$  out of  $\Pi$ 
    (3)  $\Omega = \text{crossover}(\Phi_i \text{ and } \Phi_m)$ 
    (4) if random(0,100) < mutation.frequency
        then mutate offspring with  $\Omega = \mu(\Omega)$ 
    endif
    (5) if  $\epsilon(\Omega) > \epsilon(\text{local.best})$ 
        then
            . select individual from  $\Pi_{\text{int}}$  and eliminate it
            .  $\Pi_{\text{int}} = \Pi_{\text{int}} \cup \Omega$ 
            . local.best =  $\Omega$ 
            . send local.best to all neighbored processors
        else if  $\Omega$  is selected to be included into
            then
                . select individual from  $\Pi_{\text{int}}$  and eliminate it
                .  $\Pi_{\text{int}} = \Pi_{\text{int}} \cup \Omega$ 
            endif
        endif
    (6) if neighbor  $i$  has sent new individual
        then
            . remove individual sent last by  $i$  from  $\Pi_{\text{ext}}$ 
            /* 从  $\Pi_{\text{ext}}$  中移去 */
            .  $\Pi_{\text{ext}} = \Pi_{\text{ext}} \cup \Omega_i$ 
        endif
    endwhile

```

其中, μ 是变异控制参数, ϵ 是选择控制参数。

因为本地最佳个体的分布显然是迁入式模型的主要设计原则之一,所以处理器网络的拓扑结构应该保持较小的直径,即任意两个处理器间的最大距离应该小一些。为此,采用大小合适的扩展 De Bruijn 拓扑作为互连网络。

5.6.2 迁出式算法

迁出式(emigration)和迁入式都属于孤岛模型,它们的差别仅在于交换个体的方式不同。迁入式是将最佳个体的副本(copies)同时发送给若干个邻近处理器;而迁出式则是将最佳个体的原本(originals)只发送给一个邻近处理器而自己不留备份,这种保守的迁移策略的目的是防止局部群体中出现许多的相同个体,从而有效地保持本地子群体中遗传基因的多样性。

因为在邻域内个体的迁移量很少,所以在迁出式中接收到的个体被和本地产生的个体一样对待。另外,与迁入式不同的是,这里每代产生的个体数固定为局部群体大小的一半。

迁出式遗传算法新增了两个参数来控制每个处理器上的迁移活动:send.rate 给出了向邻域迁移个体的频率,即如果 send.rate 等于 i ,则每隔 i 代迁出一次;send.best 则给出了每个迁移步发送的个体数量,即如果 send.best 等于 i ($i \geq 1$),则当前群体的前 i 个最佳个体被迁出到邻近处理器上去。

每个处理器有一个局部群体 Π_{int} , $|\Pi_{int}| = n$, $\Pi_{int} = \Phi_1, \dots, \Phi_n$ 以及集合 $\Pi_o = \Omega_1, \dots, \Omega_{max}$,它用以暂存每一代中产生的所有后代。迁出式 GA 在每个处理器上的算法可描述如下:

```

generation = 0
initialize local population  $\Pi_{int} = \Phi_1, \dots, \Phi_n$ 
while generation < max.generations do
    (1) generation = generation + 1
    (2)  $i = 1$ ;  $\Pi_o = \emptyset$                                      /* i = offspring counter */
    (3) while  $i \leq \text{max. offsprings}$  do

```

```

. if individual  $\Phi'$  was received
  then  $\Omega_i = \Phi'$ 
  else -- select  $\Phi_f$  and  $\Phi_m$  out of  $\Pi_{int}$ 
    -- if  $\text{random}(0,100) < \text{mutation.frequency}$ 
      then mutate  $\Phi_f$  with  $\Omega_i = \mu(\Phi_m)$ 
      else  $\Omega_i = \text{crossover}(\Phi_f, \Phi_m)$ 
    endif
  endif
.  $\Pi_\Omega = \Pi_\Omega \cup \Omega_i$ 
.  $i = i + 1$ 
endwhile
(4)if (generation MOD send.rate) = 0
  then for j = 0 to send.best do
    . select best individual  $\Phi_b$  from  $\Pi_{int} \cup \Pi_\Omega$ 
    . remove  $\Phi_b$  from  $\Pi_{int} \cup \Pi_\Omega$ 
    . send  $\Phi_b$  to an arbitrary selected neighbor
  endfor
endif
(5) $\Pi_{int} = \text{select the } n \text{ best individuals from } \Pi_{int} \cup \Pi_\Omega$ 
endwhile

```

与迁入式不同,迁出式意在保持本地子群体的基因控制权,为此,子群体间的通信也比迁入式少。然而,强化子群体的隔离度不应只靠减少通信来达到,还可以用大直径的网络拓扑结构和有限覆盖度的邻域结构来支持。所以他们采用了如图5.4所示的梯状拓扑。这种拓扑结构扩散性很好,而且充分利用了 Transputer 的四条通信链。在实验中,他们将邻域定义为与本处理器有直接链路连接的所有处理器。

5.6.3 扩散式算法

对邻域群体中个体的不受限的存取是并行遗传算法扩散模型的基本推广,扩散意味着持续漫延,逻辑上可以认为所有个体都在邻域

内移动,所以没有显式的个体交换,而且不同的处理器可以同时访问相同的个体。

这种算法的实现最好采用带有全局内存(用来存储子群体)的分布式系统,不幸的是,Transputer 系统没有这种能力(实际上确实没有带全局内存的 MIMD 机器存在)。处理器间的通信是交换信息的唯一途径,所以,想要得到一个其它处理器上的个体通常要引起某些通信延迟才能完成。为了提高效率,应该避免这种空闲时间。另外,因为扩散式算法中的选择操作的对象也包括所有邻近个体,而每种非随机选择策略都需要了解这些个体的适应度值,因此也导致通信开销的增加。所以,他们决定在实现扩散式算法时,在每个处理器上保存一份各邻近群体的备份。

这样,每个处理器上除了有一个局部群体 Π_{int} 之外,还保存有它的每个近邻的局部群体,这使得该处理器对这些个体的持久性存取成为可能。设一个处理器有 d 个近邻,而 Π_{Nbj} 是处理器 j 的局部群体。事实上,每个处理器的工作对象是由 Π_{int} 和 d 个 Π_{Nbj} 的备份构成的群体。但每个处理器只许用其后代去更新 Π_{int} ,所有的 Π_{Nbj} 只有在每次改变了处理器 j 的局部群体后才立即加以修改。这种方式避免了每个处理器等待从它的一个邻近者得到它所需的个体所耗费的空闲时间,而且不再需要经过通信来交换适应度值。然而,子群体备份的每次更新仍然带来额外的通信,但 Transputer 的硬件特性使这些通信对 CPU 的吞吐能力几乎没什么影响。既然邻近者个数有限,那么通信开销对并行遗传算法性能的影响应该是可以忽略的。

扩散式算法采用两个参数 degree 和 copy. size。其中 degree 决定了每个处理器的邻域的度,即邻近子群体的个数;而每个拷贝的群体的大小由参数 copy. size 决定。如果 $\text{copy. size} = |\Pi_{int}|$,则将对每个邻域群体产生一个完整的拷贝;如果 $\text{copy. size} < |\Pi_{int}|$,则只有每个邻域群体的 copy. size 个最佳个体将被拷贝。集合 Π_n 用以暂存每一代中产生的所有后代。扩散式 GA 在每个处理器上的算法可描述如下:

```

- generation = 0
initialize local population:  $\Pi_{int} = \Phi_1 \dots \Phi_n$ ,
for j=1 To degree do
    . send the copy.size best individuals of  $\Pi_{int}$  to neighbor j
    .  $\Pi_{Nbj}$ =receive copy.size individuals from neighbor j
endfor
while generation<max. generations do
    (1)generation = generation + 1
    (2)i = 1;  $\Pi_n = \emptyset$  /* i=offspring counter */
    (3)while i≤max. offsprings do
        . select  $\Phi_f$  and  $\Phi_m$  out of  $\Pi_{int} \cup \bigcup_{j=0}^{degree} \Pi_{Nbj}$ 
        . if random(0,100)<mutation. frequency
            then mutate  $\Phi_f$  with  $\Omega_i = \mu(\Phi_f)$ 
            else  $\Omega_i = \text{crossover}(\Phi_f, \Phi_m)$ 
        endif
        .  $\Pi_n = \Pi_n \cup \Omega_i$ 
        . i = i + 1
    endwhile
    (4) $\Pi_{int}$  = select the n best individuals from  $\Pi_{int} \cup \Pi_n$ 
    (5)if any of  $\Pi_{int}$ 's copy.size best individuals has changed
        then for j=1 to degree do
            send  $\Pi_{int}$ 's copy.size best individuals to neighbor j
        endfor
    endif
    (6)if neighbor j transmits individuals
        then  $\Pi_{Nbj}$ =receive copy.size individuals from neighbor j
    endif
endwhile

```

在具体实现中,他们也采用了如图5.4所示的梯状拓扑结构。

5.7 LCS 的并行实现

LCS(Learning Classifier System)是机器学习的一种方式,它结

合了子符号和产生系统两方面的特性,源于 Holland 在自适应系统方面的开拓性工作,并因为一些成功的应用而正在国际上越来越受欢迎。这些系统本质上所具备的并发性使它们很容易被映射到并行体系结构上去。本节我们将介绍基本模型并给出相应的算法。

如图 5.8 所示,LCS 主要由三个相互作用的子系统构成,而它作为一个整体又与环境相互作用着:它通过输入信息来感知环境的变化,通过动作作用于环境并接受奖励或惩罚作为反馈。执行子系统是一种产生系统,它允许多条规则同时点火;信用分配子系统的任务是评估各条规则的有用度;规则发现算法的任务是寻找新的可用规则。系统通过分类器系统和信用分配系统间的相互作用来测试新产生的规则,若可用则存活,否则由 GA 重新产生新规则。下面我们将详细介绍这三个模块是如何工作及相互作用的。

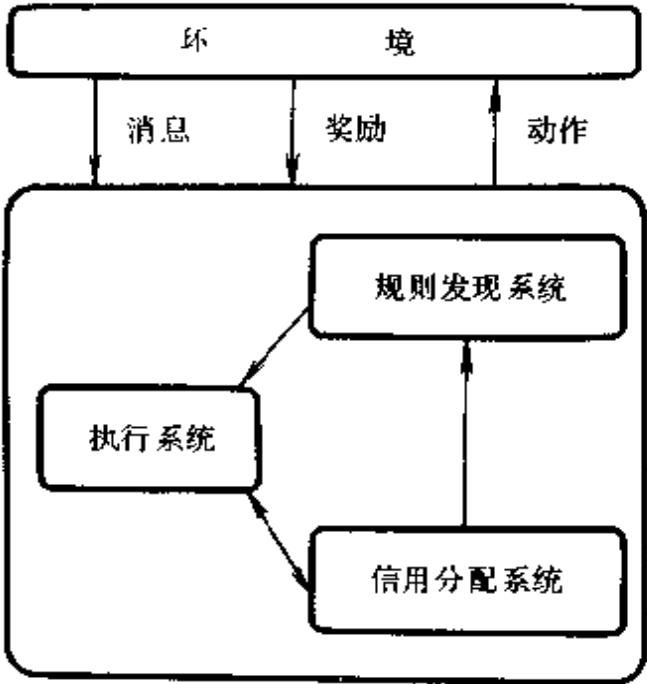


图 5.8 LCS 的功能结构

5.7.1 执行系统

分类器(classifier)也叫规则(rule)是由三个染色体构成的串(一个染色体是一个长为 n 的串),其中前两个是条件,第三个是消息/动作。当有消息能匹配某条规则的两个条件时该规则被点火,如果第三个染色体是消息,则将它附加到消息表中去;否则(即第三个染色体是动作)将它发送给效应器(effector)。

如图 5.9 所示,执行系统的主要组成如下:

- (1) 规则集合,也叫分类器集。

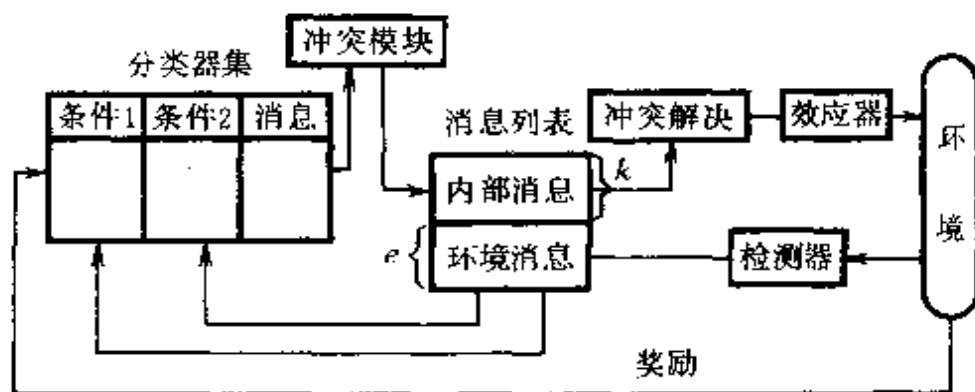


图 5.9 执行系统

(2) $k+e$ 维的消息表, 消息分两种: k 条内部消息(来自分类器集)和 e 条环境消息(来自环境)。

(3) 输入界面即探测器(detector)和输出界面即效应器(effector),它们负责系统与环境的输入输出。

(4) 模式匹配模块和冲突解决模块,它们负责在每次循环中判定哪些规则是活动的,其中又有哪些将真正被点火。

执行系统的算法可描述如下:

(1) 系统初始化(随机产生一个规则集)。

(2) 读进环境消息并把它们附加到消息表中去。

(3) 如果消息表中的消息与分类器中的两个条件相匹配, 则将该分类器状态设为活动的, 然后清除消息表。

(4) If 活动规则的个数 $\leq k$

then 将它们的消息附加到消息表中;

else 调用拍卖模块,它取所有的活动规则为输入,返回 k 条有权提交消息的规则。然后这 k 条获胜的规则附加到消息表中。

(5) 从消息表中选出待发送给效应器的消息。

(6) 调用冲突解决模块,使效应器消息彼此竞争,由获胜者决定要执行什么动作,然后它将收到与此动作的有用度成正比的奖励。

(7) 将所有分类器设置为不活动的状态。

为了能正确地工作,动作模块和冲突解决模块必须了解竞争规

则的有用度,只有用这些信息它们才能决定哪些规则有权去附加消息,或者在遇到不一致的动作要求时能据此作出选择。

5.7.2 信用系统中的分配策略

信用算法中的分配的主要任务是根据有用度值将规则分类,工作过程可描述如下:每个分类器 C 都有一个随时间变化的实数值叫做强度(strength),初始时所有分类器的强度都相同,当一个外部分类器产生了一个作用于环境的动作时,就产生一个报酬(payoff),其值取决于该动作对完成系统的目标起多大作用。这个报酬随后被传送到导致该外部分类器被点火的那个内部分类器,这种回传机制使分类器的强度值得以及时改变,从而反映出它们在整个系统完成其目标中的重要性。

实现这一策略的传统算法是桶队(bucket brigade)算法,它把分类器系统模拟成一个经济社会,每个分类器要付出一定强度才能获取向消息表附加消息的特权,并可收到来自所有活动的分类器的报酬(payments),这种报酬来源于它在上一步向消息表中附加的消息。这样,报酬通过一系列分类器从环境中来又流回到环境中去,其结果是分类器通过其产生的动作获得高额奖励,从而强度得到了提高。

桶队算法是在上述执行系统算法基础上对拍卖模块和冲突解决模块进行了改造:在拍卖步,所有的活动规则根据其强度提出一个报价,然后选出报价最高的 k 个去传递消息;在冲突解决步,传递了消息的分类器以及导致它活动的分类器通过回传机制修改强度值,然后再根据强度去竞争效应器。

有关桶队算法,读者可参阅第四章的有关内容。

5.7.3 遗传算法在 LCS 中的应用

在 LCS 中应用的遗传算法与优化问题中的有所不同。实际上,为了维持系统的性能,只有少量规则被新规则取代。遗传算法以分类

器强度作为适应度,仅当桶队算法达到稳定状态时遗传算法应用于分类器集才有用。因此,遗传算法用得很少,通常每隔1000~10000个桶队步才调用一次。下面我们描述一下调用一次遗传算法的步骤:

- (1) 取分类器集合为初始群体 P ;
- (2) 根据适应度大小(即分类器的强度)给 P 中的个体排序;
- (3) 选适应度最小的 $2K$ 个个体准备被替代;
- (4) 选适应度最大的 $2K$ 个个体复制作父母;
- (5) 将(4)步选出的个体配对,从而得到 k 对有用的规则;
- (6) 将 k 对规则进行交叉操作;
- (7) 对所得的 $2k$ 个个体进行变异操作;
- (8) 用新的 $2k$ 个个体取代(1.2)步选出的 $2k$ 个无用个体。

5.7.4 LCS 的一个 MIMD 实现

如前所述,在 LCS 的每次循环中规则的点火是基于消息表中的消息的,所以常常会发生许多分类器同时活动的情形,这种大的并发激活的可能性,就要求硬件系统为匹配和拍卖步提供大的计算能力。很显然,我们可以将这些计算分布到一组处理单元上,使之并行工作,每个处理器可负责一小组分类器与消息表中的消息的匹配工作。

1. 低级并行化

我们发现在每个分类器中匹配和消息产生是可以相互独立进行的,所以自然可以将分类器进程 CF 分成一系列子进程 $CF_i, i=1, \dots, n$, 每个子进程负责分类器集合的一部分工作。 n 越大并发性越高,当 n 等于 CF 的个数时,每个子进程 CF_i 只负责一个分类器。在 Dorigo 和 Maniezzo 的基于 Transputer 的实现中,他们给每个处理器分配100~500条规则^[23], CF_i 集可以组成一种层状结构,例如树或环状网络,所选的结构对计算负载的分配有重要影响^[24]。

低层次并行化是将标准的 LCS 以一种中央驱动的分布式算法来实现,整个 LCS 系统被分布在一个主从网络上。为了提高 CPU 利用率,可以在主处理器上也分布一个 CF_i 进程。另外,还有与宿主程

序的界面,它负责感应器和效应器的工作。

例如,在连成流水线结构的9片 Transputer 上,系统可以分布成二叉树形式(图5.10)或双流水线结构(图5.11),但前者将导致从根处理器到每个从处理器没有直接的硬链连接,从而通信步很长。又

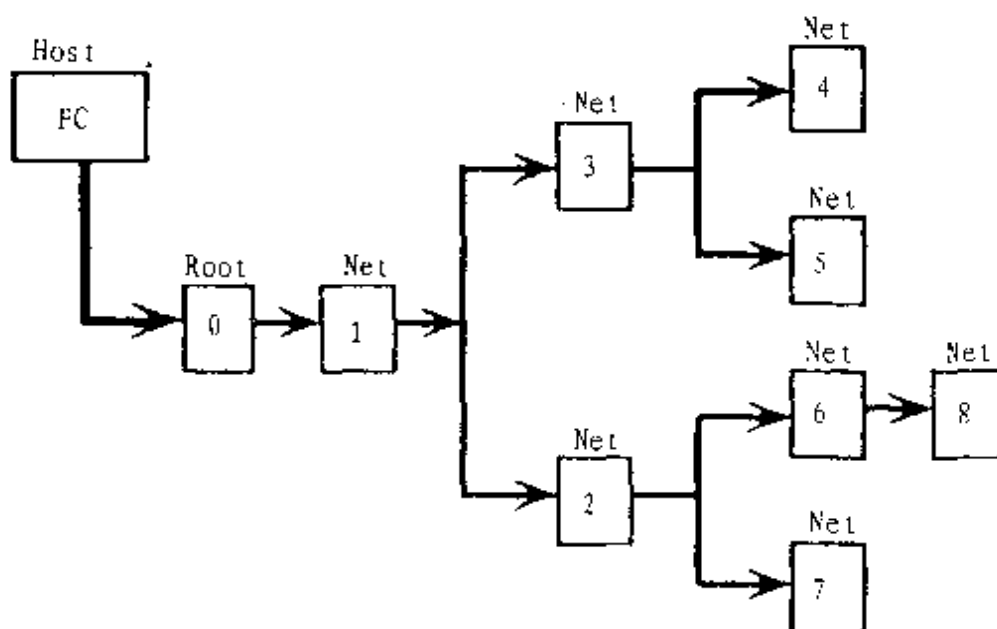


图 5.10 二叉树结构的并行 LCS

因为并行 LCS 的工作数据分散在处理器网中,还要沿着同样的路径

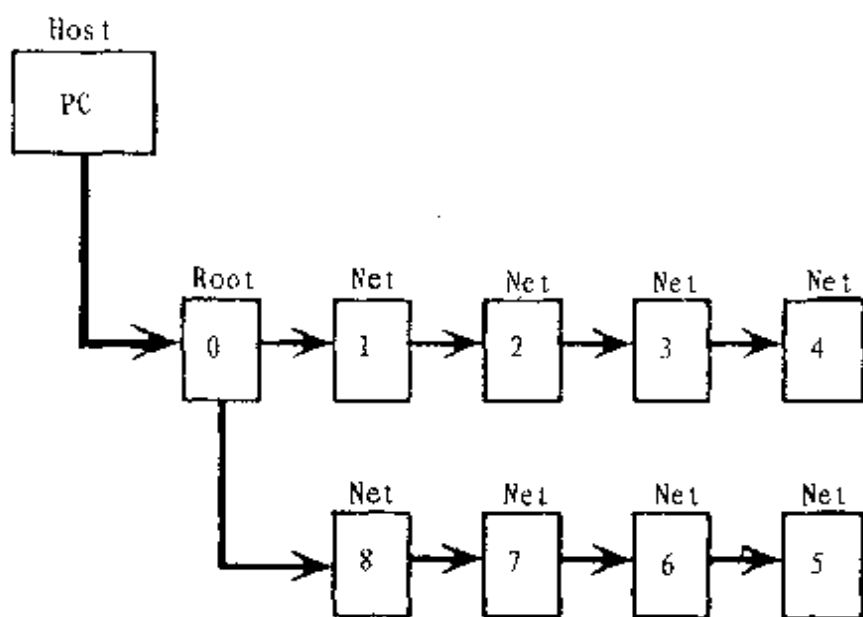


图 5.11 双流水线结构的并行 LCS

将结果收集上来,所以通信路径的长短对系统的影响很大。因此,采用二叉树结构效率不高。

2. 高级并行化

当 LCS 被应用于多目的问题时,低层次并行化就显得不足了,不幸的是大多数实际问题都是多目的的。传统的 LCS 对于求解多目标任务有困难,因为它们没有处理不同规则集的清晰的机制(即每个规则集用于求解一个不同的目的),似乎它们需要使用一些更高的技术来处理由多样性所造成的增高的复杂度。因此,虽然它较快而且能控制较大的规则集。但低级并行化 LCS 仍然不够实用。另外,在低级并行化 LCS 中还存在可缩放性的问题:网络增大会使通信负担增大,而在计算速度方面所获增益越来越小,趋向于一个渐近的极限。

处理复杂问题的一个更好的办法是把它看成是一组简单问题,其中每个都可由较小的 LCS 来解决^[25,26]。我们可以将处理器网络划分为各有一定的大小和拓扑结构的子集,每个子集上分配一个以低级并行化方式分布的 LCS。每个 LCS 根据它所收到的输入解决一个特定的子目标。而且通过它自己的监测器来感知外部环境,而输出界面则是所有子 LCS 共享的,所以各 LCS 所提交的动作之间会有一些相互作用。这种模式可以以许多不同的方式来构成,也为构造学习系统提供了一种工具。

5.7.5 LCS 在 CM 机器上的实现

在此我们将讨论分布式遗传算法和分类器系统在连接机器(CM)上的实现及其结果。所用的 CM 包含有64K(65536)个处理器,每个处理器上有1M 局部内存。每个操作都是以高于1000MIPS 的速度在每个处理器上并发执行。如果数据元素多于处理器个数,则可通过划分每个物理处理器的内存来虚拟地实现它。显然,这时物理处理器个数越多,应用程序运行得越快。所有的处理器通过选路器相互连接,选路器是一种高速通信设备,负责处理器间的并行通信,这种机制使应用程序可以动态地重构通信拓扑。这样的硬件结构已被一些

研究者用来并行实现 GA 和 CS。

Robertson^[27,28]在 CM 上实现了他的 CFS-C 分类器系统环境,称为 LCS,系统的速度几乎与所用分类器的个数无关,从而为在大任务领域评价 LCS 和 GA 提供了一条途径。

*LCS 应用于实际问题时,采用了大量分类器以便有效地搜索规则空间,采用了有限的消息表长度以便促进活动的分类器之间的竞争。不论分类器个数有多少,整个匹配进程执行时间为3.5ms,为处理消息表中的每条消息耗时0.5ms。

*CFS 的规则发现系统是由遗传算法来构成的,每隔11次循环步执行一次,每次替换掉群体中5%的个体,但 Robertson 没有采用细粒度模型而是尝试去将标准的串行遗传算法并行化。

在*CFS 中,遗传算法的三个基本操作被实现如下:

选择是通过“并行随机加权选择”算法实现的,它给每个处理器产生一个零到相关权值之间的随机数,随后分类器按这种数排序,低端的5%个体被高端的5%个体取代,这一过程所用时间为23ms。

交叉在替代的分类器上进行。在有序的分类器序列中,将奇偶数的分类器施行配对。为每对产生一个随机数,并执行标准的单点交叉。每个新产生的分类器的强度都设置为其父母强度和群体平均强度的均值。

变异是按要变异的基因个数的波松分布进行操作的。概率的分布以表的形式存储,一旦决定了个数,实际要改变的基因按均衡的概率进行挑选。

这样的遗传算法运行在65536个分类器上,总的执行时间为400ms,它可用来作字母序列的预测^[29]。

Collins 和 Jefferson 也在 CM 上实现了遗传算法^[30],他们用 CM 上的遗传算法求解图的划分问题,采用细粒度模型去分布算法并将之与标准的遗传算法作比较,结果再次显示局部处理机制使遗传算法更快,鲁棒性也更好,对于64个顶点的图的测试问题总能找到最佳解,而标准遗传算法在相当一部分运行中不能找到最佳解。

参 考 文 献

- [1] Holland J H. Outline for a Logical Theory of Adaptive Systems, Journal of the Association for Computing Machinery, 1962(3):297~314
- [2] Pettey C B, Leutze M R, Grefenstette J J. A Parallel Genetic Algorithm Proc of the second ICGA, 1987, 155~161
- [3] Cohoon J P, Hegde S U, Martin W N, D S. Richards. Punctuated Equilibria: A Parallel Genetic Algorithm. Proc of the second ICGA, 1987, 148~154
- [4] Tanese R. Parallel Genetic Algorithm for a Hypercube. Proc of the second ICGA, 1987, 177~183
- [5] East I R, Macfarlane D. Implementation in Occam of Parallel Genetic Algorithms on Transputer Networks. Parallel Genetic Algorithms: Theory and Applications, J Stender, Ed., IOS Press, 1993, 43~63
- [6] Pettey C C, Leutze M R. A Theoretical Investigation of a Parallel Genetic Algorithm. Proc of the third ICGA, 1989, 398~405
- [7] Brown D E, Huntley C L, Spillane A R. A Parallel Genetic Heuristic for the Quadratic Assignment Problem. Proc of the third ICGA, 1989, 406~415
- [8] Kroger B, Schwenderling P, Vornberger O. Parallel Genetic Packing of Rectangles. Proc of the first int wks on Parallel Problem Solving from Nature (PPSN-90), 1990, 8~11
- [9] Cohoon J P, Martin W N, Richards D S. Genetic Algorithms and Punctuated Equilibria. Proc of the first int wks on Parallel Problem Solving from Nature (PPSN-90), 1990, A~21

- [10] Cohoon J P, Martin W N, D S Richards. A Multi-population Genetic Algorithm for Solving the K-Partition Problem on Hyper-cubes. In: Proc of the fourth ICGA, 1991, 244~248
- [11] Muhlenbein H, Schomisch M, Born J. The Parallel Genetic Algorithm as Function Optimizer. Proc of the fourth ICGA, 1991, 271~278
- [12] Prahlada B B and Hansdah R C. Extended Distributed Genetic Algorithm for Channel Routing. IEEE Trans on Neural Networks, 1993, 726~733
- [13] Wilson S L. Classifier Systems and the Animat Problem. Machine Learning, 1987, 2(3):199~228
- [14] Sannier A V, Goodman E D. Genetic Learning Procedures in Distributed Environments. Proc of the second ICGA, 1987, 162~169
- [15] Manderick B, Spiessens P. Fine-grained Parallel Genetic Algorithms. Proc of the third ICGA, 1989, 1989, 428~433.
- [16] Muhlenbein H, Gorges-Schleuter M and Kramer O. Evolution Algorithms in Combinatorial Optimization. Parallel Computing, 1988, 7(1):65~88
- [17] Muhlenbein H. Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization. Proc of the third ICGA, 1989, 416~421
- [18] Gorges-Schleuter M. ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy. Proc of the third ICGA, 1989, 422~427
- [19] Gorges-Schleuter M. Explicit Parallelism of Genetic Algorithms through Population Structures. Proc of the first int wks on Parallel Problem Solving from Nature (PPSN-90), 1990, A-6

- [20] von Laszewski G and Muhlenbein H. A Parallel Genetic Algorithm for the Graph Partitioning Problem. Proc of the first int wks on Parallel Problem Solving from Nature (PPSN — 90, 1990,B—12
- [21] GTalbi E and Bessiere P. A Parallel Genetic Algorithm for the Graph Partitioning Problem. Proc ACM—ICS' 91,1991
- [22] Kroger B, Schwenderling P and Vornberger O. Parallel Genetic Packing on Transputers. Parallel Genetic Algorithms: Theory and Applications, Stender J, Ed. IOS Press,1993,151 ~185
- [23] Dorigo M. Using Transputer to Increase Speed and Flexibility of Genetics-based Machine Learning Systems. Microprocessing and Microprogramming, Euromicro Journal, North Holland(1992)
- [24] Camilli A, Di Meglio R, Baiardi F, Vanneschi M, Montanari D and Serra R. Classifier System Parallelization on MIMD Architectures. Tech Rept 3/17 CNR March 1990
- [25] Dorigo M and Schnepf U. Genetics-based Machine Learning and Behavior Based Robotics: A New Synthesis. Tech Rept 91 — 44 Politecnico di Milano—Dipartimento di Elettronica — PM— AI &R Project, Italy (To appear on IEEE Trans on Systems, Man, and Cybernetics), February, 1991
- [26] Dorigo M and Schnepf U Organisation of Robot Behavior Through Genetic Learning Processes. Proc of ICAR — Fifth International Conference on Advanced Robotics, IEEE, Pisa, Italy, June 1991
- [27] Robertson G. Parallel Implementation of Genetic Algorithms in a Classifier System. Proc of the second ICGA, 1987,140~147

- [28] Robertson G, Parallel Implementation of Genetic Algorithms in a Classifier System. Proc of the second ICGA, Lawrence Erlbaum, MIT—Cambridge—MA, 1987
- [29] Robertrtson G and Riolo R L. A Tale of Two Classifier Systems. Machine Learning 1988,3, 139~159
- [30] Collins R J and Jefferson D R. Selection in Massively Parallel Genetic Algorithms. Proc of the fourth ICGA, 1991,249~256

第六章 神经网络、模糊集理论和进化算法

神经网络、模糊集理论和以遗传算法为代表的进化算法都是仿效生物处理模式以获得智能信息处理功能的理论。其中，神经网络着眼于脑的微观网络结构，通过大量神经元的复杂连接，采用由底到顶的方法，通过自学习、自组织和非线性动力学所形成的并行分布方式，来处理难于语言化的模式信息。模糊集理论则着眼于可用语言和概念作为代表的脑的宏观功能，使用由顶到底的方法，按照人为引入的隶属度函数和串并行规则，逻辑地处理包含有模糊性的语言信息。而进化算法则是模拟生物的进化现象（自然淘汰、交叉、变异等），并采用自然进化机制来表现复杂现象的一种概率搜索方法，以达到快速有效地解决各种困难问题。

神经网络、模糊集理论和进化算法三者目标相近而方法各异。因此，将它们相互结合，必能达到取长补短的作用。近年来，在这方面已经取得不少研究成果，并形成了“计算智能”的研究领域。

本章第一节介绍遗传算法与神经网络相结合的进化神经网络；第二节介绍遗传算法与模糊集理论相结合并将它应用于模式识别；第三节介绍在统一描述框架下的进化算法的三个典型算法：遗传算法、进化规划和进化策略。

6.1 遗传算法与神经网络

6.1.1 神经网络的发展

神经网络(NN)系统理论是近年来人工智能的一个前沿研究领域。与以往的基于符号机制的冯·诺伊曼计算机理论迥然不同，神经

网络是基于连接机制的大规模并行处理和分布式的信息存储,它依靠大量神经元的联接以及这种连接所引起的神经元的不同兴奋状态和系统所表现出的总体行为。与当今计算机相比,更加接近人脑的信息处理模式^[1]。

自1943年第一个神经网络模型——MP 模型被提出至今,神经网络的发展十分迅速,特别是1982年提出的 Hopfield 神经网络模型和 Rumelhart 于1985年提出的反向传播算法(BP),使 Hopfield 模型和多层前馈型神经网络成为用途广泛的神经网络模型,在语言识别,模式识别,图像处理和工业控制等领域颇有成效。

1. 神经网络简介

(1) 基本结构。

神经网络是由大量神经元广泛互连而成的复杂网络系统。单一神经元可以有許多输入、输出。神经元之间的相互作用通过连接的权值体现。神经元的输出是其输入的函数。常用的函数类型有:线性函数, S 型函数和阈值型函数。虽然单个神经元的结构和功能极其简单和有限,但大量神经元构成的网络系统的行为是极其丰富多采的。图6.1表示出单个神经元和 Hopfield 模型的结构。

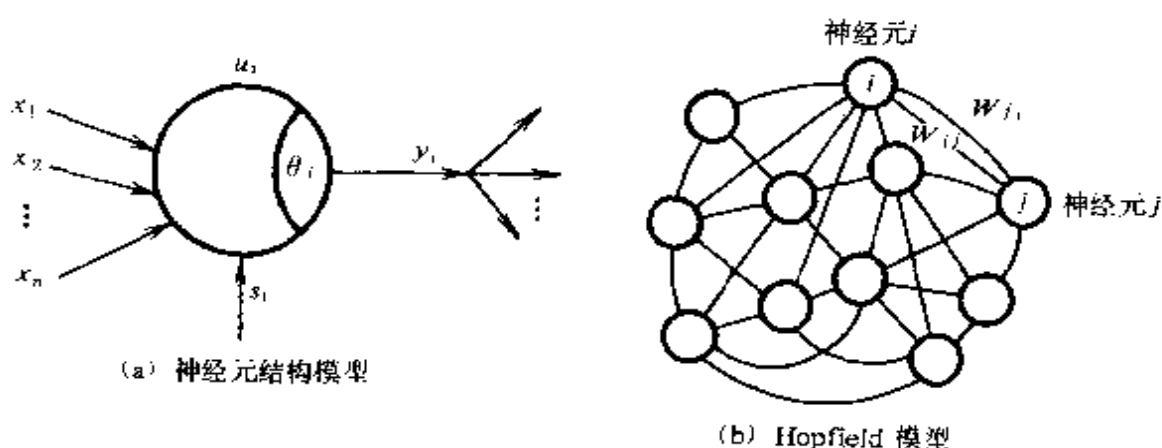


图 6.1 神经元模型和 Hopfield 模型

在图6.1(a)中, u_i 为神经元的内部状态, θ_i 为阈值, x_i 为输入信号, w_{ij} 表示从 u_j 到 u_i 连接的权值, s_i 表示外部输入信号,则神经元的输入为 $net_i = \sum_j w_{ij}x_j + s_i - \theta_i$,输出为 $Y_i = f(net_i)$,其中 f 是神

经元的转换函数。

在图6.1(b)中,Hopfield 模型是由许多神经元构成的互连网络,适当选取神经元兴奋模式的初始状态,则网络的状态将逐渐到达一个极小点(稳定点),从而可以联想到稳定点处的样本。神经网络的基本特征是:

- 大规模并行处理:神经网络能同时处理与决策有关的因素,虽然单个神经元的动作速度不快,但网络的总体处理速度极快。

- 容错性:由于神经网络包含的信息是分布存储的,即使网络某些单元和连接有缺陷,它仍可以通过联想得到全部或大部分信息。

- 自适应性和自组织性:神经网络系统可以通过学习不断适应环境,增加知识的容量。

(2) 学习规则。

学习规则决定了神经网络的连接权值的变化,典型学习规则有:

- Hebb 规则,如 Hopfield 网络采用的修正 Hebb 规则为:

$$\Delta W_{ij} = (2a_i - 1)(2a_j - 1)$$

其中 a_i, a_j 分别是节点 i 和 j 的激活值。常用于自联想网络。

- δ 学习规则和广义 δ 学习规则:可用于学习非线性可分函数。BP 网络就用到这些学习规则。

- 模拟退火:是 Boltzmann 机的学习规则,它能学习非常复杂的非线性可分函数。

- 无教师学习规则:它利用自适应学习方法,使节点能选择接收输入空间上的不同特性。

(3) 工作方式。

- 前向式(或称前馈式):在这种工作方式中,网络被分为输入层、隐层和输出层,信息从输入层开始,经由隐层流向输出层,如图6.2所示。

典型的前向式网络如“感知器”和 BP。

- 演化式:此时输入层和输出层合二为一,例如 Hopfield 网络。

当前,神经网络理论与应用都取得了长足的进步,其中,多层前

馈型神经网络 BP 是用途最广泛的网络之一,虽然如此,神经网络理论还存在着许多缺陷,例如训练速度慢,易陷入局域极小和全局搜索能力弱等。

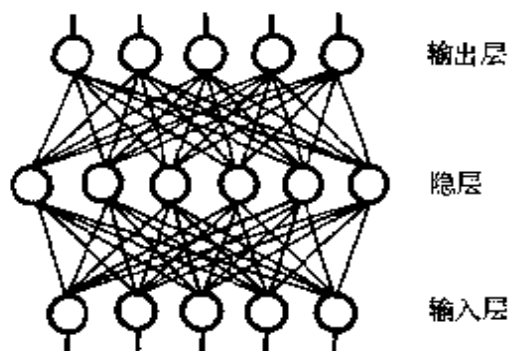


图 6.2 前向式神经网络的结构与工作方式

2. 遗传算法与神经网络

遗传算法(GA)的出现使神经网络的训练有了一个崭新的面貌,目标函数既不要求连续,也不要求可微,仅要求该问题可计算,而且它的搜索始终遍及整个解空间,因此容易得到全局最优解。本节将以训练 BP 网络为例来说明用遗传算法优化神经网络的思想与方法,其它神经网络的优化也可类似处理。

BP 模型是目前应用最广泛的一种学习算法,在 PDP 小组提出的 BP 模型中,网络分为输入层、隐层和输出层。隐层可以有一层或多层,节点的作用函数选 S 型函数: $f(x) = \frac{1}{1+e^{-x}}$ 或 $f(x) = \frac{1}{2}(1 + \tanh(\sigma/x_0))$ 其中 σ 为对应节点的输入。

输出节点和隐节点的输出为:

$$\begin{aligned}\delta_{pj} &= (T_{pj} - O_{pj}) \cdot f'(\text{net}_{pj}) \\ \delta_{pi} &= f'(\text{net}_{pi}) \cdot \sum_k (\delta_{pk} W_{ki})\end{aligned}$$

算法学习过程如图 6.3 所示。

用遗传算法优化神经网络,可以使得神经网络具有自进化、自适应能力,从而构造出进化的神经网络(ENN)^[2],它主要包括三个方面:1)连接权的进化;2)网络结构的进化;3)学习规则的进化。

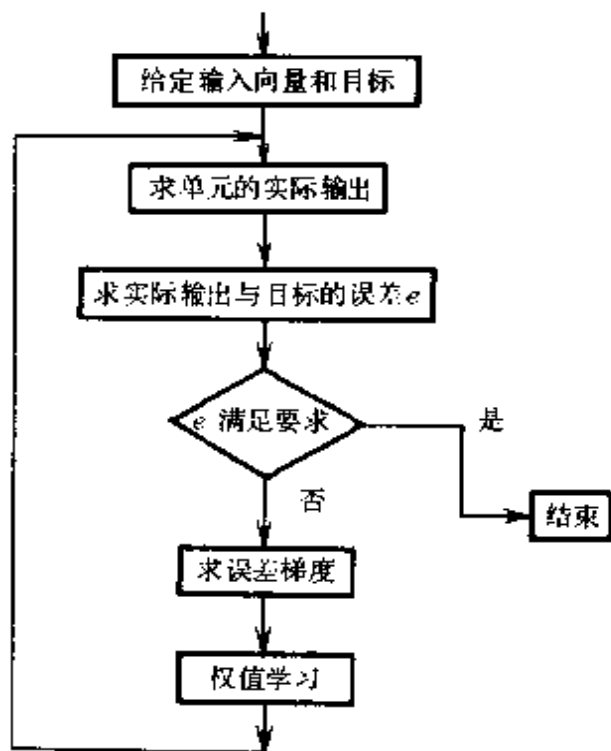


图 6.3 BP 网络学习过程

6.1.2 神经网络连接权的进化

神经网络连接权的整体分布包含着神经网络系统的全部知识,传统的权值获取方法都是采用某个确定的权值变化规则,在训练中逐步调整,最终得到一个较好的权值分布,BP 网络的学习过程正是如此。这就可能由于算法的缺陷和导致不满足问题的要求,如训练时间过长,甚至可能因陷入局域极值而得不到适当的权值分布,如果用遗传算法来优化连接权,可望解决这个问题^[3,4]。

用遗传算法优化神经网络连接权的过程如下:

(1) 随机产生一组分布,采用某种编码方案对该组中的每个权值(或阈值)进行编码,进而构造出一个个码链(每个码链代表网络的一种权值分布),在网络结构和学习规则已定的前提下,该码链就对应一个权值和阈值取特定值的一个神经网络。

(2) 对所产生的神经网络计算它的误差函数,从而确定其适应度函数值,误差越大,则适应度越小。

(3) 选择若干适应度函数值最大的个体,直接遗传给下一代。

(4) 利用交叉和变异等遗传操作算子对当前一代群体进行处理,产生下一代群体。

(5) 重复(2),(3),(4),使初始确定的一组权值分布得到不断的进化,直到训练目标得到满足为止。

1. 编码方案

对网络中的权值和阈值进行编码主要有两种方法:一是采用二进制编码方案;另一是实数编码方案,有关编码方法详见第二章第六节。

(1) 二进制编码方案。

二进制编码是最常见的一种编码方案,在这种方案中,每个权值都用一定长的 0/1串表示,阈值被看作是输入为-1的连接权,例如,若所有的权值都在-127~+128之间,则可以用8位0/1串完全表示。假设网络的连接性质已经确定,而每个连接权均在某一预先定义的限定范围内变化,那么各连接权的字符串表示值和实际权值之间有如下关系:

$$W_i(i, j) = W_{\min}(i, j) + \frac{\text{binreplace}(t)}{2^l - 1} [W_{\max}(i, j) - W_{\min}(i, j) + 1]$$

其中,binreplace(t)是由 l 位字符串所表示的二进制整数, $[W_{\min}(i, j), W_{\max}(i, j)]$ 为各连接权的变化范围。然后将所有权值对应的0/1串级联在一起,得到的一个很长的二进制字符串就代表着网络的一种权值分布。表6.1 给出了二进制编码的一种方案。

表 6.1 8位二进制数编码方案

| 权值 | 二进制编码 | 权值 | 二进制编码 |
|-----|----------|----|----------|
| 127 | 00000000 | 0 | 01111111 |
| 126 | 00000001 | 1 | 10000000 |

续表

| 权值 | 二进制编码 | 权值 | 二进制编码 |
|------|----------|-----|----------|
| -125 | 00000010 | . | . |
| -124 | 00000011 | . | . |
| . | . | . | . |
| . | . | 125 | 11111100 |
| . | . | 126 | 11111101 |
| -2 | 01111101 | 127 | 11111110 |
| -1 | 01111110 | 128 | 11111111 |

例6.1 XOR 问题的网络连接权编码。

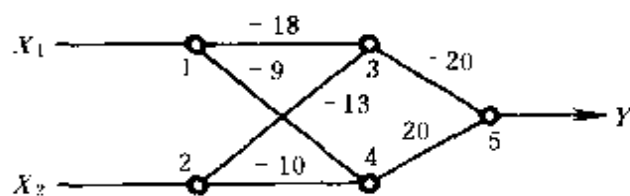


图 6.4 XOR 问题的一种权值分布

XOR 问题是神经网络中的一个著名例子，其输入到输出的映射是线性不可分的，可描述为：

$$Y = f(x_1, x_2) = \begin{cases} 0, & x_1 = 0, & x_2 = 0 \\ 1, & x_1 = 0, & x_2 = 1 \\ 1, & x_1 = 1, & x_2 = 0 \\ 0, & x_1 = 1, & x_2 = 1 \end{cases}$$

设 XOR 问题的神经网络结构与权值分布如图 6.4 所示，为简化起见，假设阈值为零，则上述权值分布可以用一个 $8 \times 6 = 48$ 位长的字符串表示，即

| | | | | | |
|----------|----------|----------|----------|----------|----------|
| 01101101 | 01110010 | 01101011 | 01110110 | 01110101 | 10010011 |
| ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| -18 | -13 | -20 | -9 | -10 | 20 |

在将各权值对应的字符串级联在一起时,一种较好的级联次序是把与同一隐节点相连的连接权对应的字符串放在一起,这是因为隐节点在神经网络中起特征抽取和特征探测作用,若将其与同一隐节点相连的连接权对应的字符串分开,则将增加获取特征的难度,因为遗传操作算子很容易破坏这些特征,上面的字符串就是按照这种方法得到的。

采用二进制编码方案时,也可以同时将粒度(即编码长度)级联在字符串上,如图6.5所示。

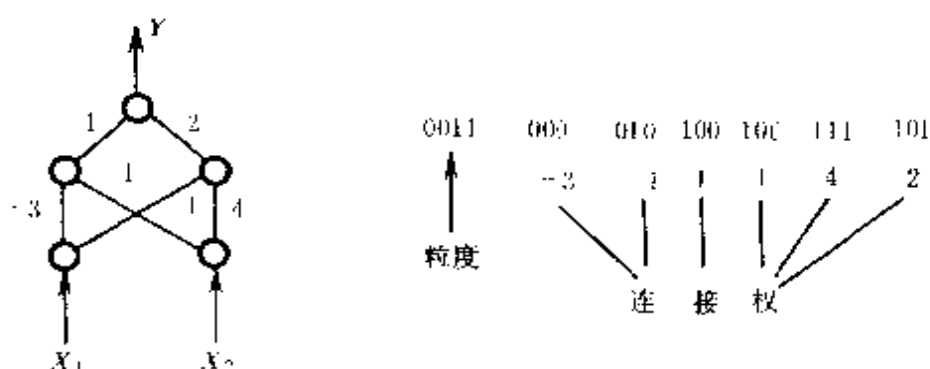


图 6.5 编码粒度和连接权同时级联在一起的字符编码

(2) 实数编码方案。

在这种方案中,每个连接权值直接用一实数表示,一个网络权值分布用一组实数来表达,遗传操作在两组实数上进行。显然,这时一般的遗传操作算子不能直接使用了,数字的改变只能通过偶尔的变异实现。

(3) 两种编码方案的特点。

二进制编码方案的优点是它非常简单、通用,一般的遗传操作算子诸如交叉和变异都可以直接使用,而不必专门设计其它复杂的操作算子,其缺点有:

- 可量测性不强,不直观;

- 精度不高:因为权值是实数型,若将它们用二进制数编码实际上是用离散值来尽量逼近权值本身,这就有可能导致因某些实数权值不能近似表达而使网络的训练失败,例如在前面的例题中:若权值

-18改为-18.4,则编码01101101实际上是一18.4的近似值;

•编码时字符串不能太长或太短:太长将导致遗传算法训练的解空间过大,算法需要花费很长时间才能得到最优解,而太短则使精度不高。

实数编码方案的优点是它非常直观,且不会出现精度不够的情况,但它的缺点也很明显,虽然对某些问题不一定都必要,但有些问题可能需要设计专门的遗传操作算子。

2. 适应度函数的确定

在前面我们已经给出了一种确定适应度函数的方法,这种方法是通过计算网络的误差,并认为误差大的网络其适应度函数值就小,从而得到适应度函数。例如可设适应度函数为: $F=C-e$,其中 C 是一常数, e 为误差(真实输出与计算输出之间的差距);又如,可设 $F=C-E$,其中 E 为能量函数。

适应度函数除了与误差函数有关外,还与进化的时间和网络的复杂度相关联,其中网络复杂度在后面的网络拓扑结构部分更显得重要。

3. 遗传操作

遗传操作的目的是利用选择,交叉和变异等遗传操作算子使由诸神经网络个体组成的种群由上一代向下一代进化,其方法和简单遗传算法中的遗传操作类似。有时也需要设计专门的操作算子。

4. 混合训练神经网络

将基于遗传算法的遗传进化和基于梯度下降的反传训练结合,就称为神经网络的混合训练,这种训练取两种方法的各自特点^[2]:

(1) 基于遗传算法的遗传进化方法可以在一复杂的、多峰的、非线性及不可微的空间中实现全局搜索,它不需要有关误差函数梯度的信息,这在很难获取这些信息的情况下具有独特的优点。另一方面,它不需考虑误差函数是否可微,从而可以在误差函数中增加某些惩罚项,以提高网络的通用性,同时降低网络的复杂度。不过,当在训练过程中容易获取梯度信息时,基于遗传算法的进化方法在速

度上就未必优于 BP 算法或其它基于梯度的训练算法了。

(2) 遗传算法和 BP 算法的结果都对训练过程中用到的算法参数很敏感, BP 算法的结果同时还依赖于网络的初始状态。

(3) 遗传算法擅长全局搜索, 而 BP 算法在用于局部搜索时显得比较有效。

所以, 将遗传算法与 BP 算法结合, 进行神经网络的混合训练是一可行的途径。首先用遗传算法方法对初始权值分布进行优化, 在解空间中定位出一个较好的搜索空间; 然后采用 BP 算法在这个小的解空间中搜索出最优解。一般情况下, 混合训练的效率 and 效果比单独用遗传算法进化方法或用 BP 训练方法的结果要好。

6.1.3 神经网络结构的进化

神经网络的结构包括网络的拓扑结构即网络的连接方式和节点转换函数两部分。结构的优劣对网络的处理能力有很大影响, 一个好的结构应能圆满解决问题, 同时不允许冗余节点和冗余连接的存在。不幸的是, 神经网络结构的设计基本上还依赖于人的经验, 尚没有一个系统的方法来设计一个适当的网络结构。目前, 人们在设计网络结构时, 或者干脆预先确定, 或者采用递增或递减的探测方法。递增式探测方法是: 从一很小的网络结构(最小数目的隐层、节点和连接权)开始, 在训练过程中, 根据特定问题的需要, 逐渐增加结构的各个部分, 直至找到能解决问题的网络结构为止; 递减式探测方法与递增式探测方法正好相反^[2, 6~8]。用遗传算法来进化神经网络结构步骤是:

- (1) 随机产生 N 个结构, 对每个结构编码, 每个编码个体对应一个结构;
- (2) 用许多不同的初始权值分布对个体集中的结构进行训练;
- (3) 根据训练的结果或其它策略确定每个个体的适应度;
- (4) 选择若干适应度值最大的个体, 直接继承给下一代;
- (5) 对当前一代群体进行交叉和变异等遗传操作, 以产生下一

代群体。

(6) 重复(2)~(5),直到当前一代群体中的某个个体(对应着一个网络结构)能满足要求为止。

一般情况下,结构即指拓扑结构,所以下面就针对网络的拓扑结构进行讨论。

1. 结构描述

根据参加编码的结构信息的多少,结构的描述方法有两种:一种是采用直接编码模式;另一种是采用间接编码模式。

(1) 直接编码模式。

在这种编码模式中,网络结构的每个连接关系都编码成二进制串。其方法是用一 $N \times N$ 的矩阵 $C = (C_{ij})_{N \times N}$ 表示一个网络的结构,其中 N 是网络的节点数, C_{ij} 的值说明网络中节点之间是否有连接, $C_{ij} = 1$ 表示两节点间存在连接, $C_{ij} = 0$ 表示两节点间没有连接。事实上,当 C_{ij} 取实数值时, C_{ij} 就完整地表示出节点 i 与节点 j 之间的连接关系,即既体现出两者之间有无连接,也说明了两节点之间连接的强度(即连接权)。这样,一个矩阵表示一个神经网络,级联矩阵的所有行(或列)所得到的一个二进制串,就对应一个神经网络结构。网络的约束(即此网络的特殊性)可以通过矩阵的特殊形式体现。例如,在前向式神经网络对应的矩阵中,只有矩阵的上三角有非零元素。

这种编码模式的优点是简单、直接,它特别适于进行小结构神经网络的进化;缺点是对大结构神经网络不适用。对大结构神经网络采用这种编码模式时,编码长度非常长,这将导致算法的搜索空间显著增大。改进的方法是利用领域知识来初始化矩阵,正象上面举例的前向式网络所示,由于前向式网络中后一层对前一层设有反馈,所以网络对应的矩阵中下三角都只能为“0”。

例6.2 XOR问题的网络结构编码。

假设同层节点之间没有连接,XOR问题对应的网络可以得到简化。图6.6所示的结构对应的矩阵为:

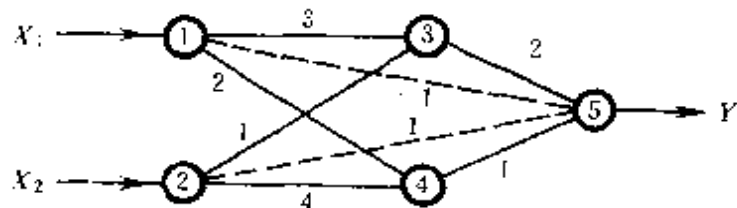


图 6.6 网络的一种可能结构

$$C_{5 \times 5} = \begin{vmatrix} 0 & 0 & 3 & 2 & -1 \\ 0 & 0 & -1 & 4 & 1 \\ 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

我们在编码时,只需考虑主对角线上面的元素。若采用三位字符编码,权值从-3~4,则编码方案为:

0011 1 110 1 010 1 101 1 111 1 010 1 100 1 001 1 100

粒度 C_{13} C_{23} C_{14} C_{24} C_{15} C_{25} C_{35} C_{45}

若进一步假设输入节点与输出节点间不直接相连,则编码方案为:

0011 1 110 1 010 1 101 1 111 0 0 1 001 1 100

粒度 C_{13} C_{23} C_{14} C_{24} C_{35} C_{45}

在上两个编码方案中,有下划线的“1”表示有连接关系存在,有下划线的“0”表示此两节点间不存在连接。

(2) 间接编码模式。

间接编码模式只编码有关结构的最重要的特性,如隐层数,每层的隐节点数,层与层之间的连接数等参数,而有关各个连接的细节留到后面讲进化规则时处理。这种编码模式可以显著缩短字符串长度,但导致了进化规则的复杂。

2. 拓扑结构与转换函数同时进化

神经网络结构包括网络的拓扑结构和节点转换函数两部分。目前,对神经网络结构进化的研究集中在网络拓扑结构的进化上,这时的节点转换函数是预先设定的。但选择一个适当的节点转换函数

也是十分重要的,如同时将网络的拓扑结构和节点转换函数进行进化处理,则效果会更好。

例6.3 用遗传算法解决 XOR 问题

1. 优化 XOR 网络的权值

假设神经网络的结构如图6.4所示。神经元的激活函数采用S型函数 $f(x)=1/2 \cdot (1+\text{th}(x/u))$ 其中 u 为一常数,学习样本共有4个,即(0,0,0),(1,1,0),(0,1,1)和(1,0,1)。

(1) 采用二进制数编码方案:由于网络中权值有 $W(1,3), W(1,4), W(2,3), W(2,4), W(3,5), W(4,5)$ 等5个,阈值有3个,即节点3,节点4,节点5的阈值,假设权值和阈值都在-30.0到+30.0之间,每个权值(或阈值)用一8位0/1串表示,则一个72位的0/1串就对应于一个神经网络。

每一个8位0/1串与对应权值的关系是:

$$W_i = (-127 + y_i)/128 \times 30.0$$

其中 y_i 是此8位0/1串对应的十进制数值。

(2) 适应度函数的确定方法很多,在本例中有四种可供选择的方案:

$$. F = C - e$$

$$. F = 1/e$$

$$. F = C - E$$

$$. F = 1/E$$

其中 C 为一常数, e 为误差:

$$e = \sum_m \sum_k (Y_{mk} - \bar{Y}_{mk})$$

E 为网络的能量函数:

$$E = \sum_m \sum_k (Y_{mk} - \bar{Y}_{mk})^2$$

Y_{mk} 及 \bar{Y}_{mk} 分别为第 m 个训练样本的第 k 个输出节点的期望输出与实际输出, e 的计算过程(即神经网络的正向处理过程),包括下面几步:

1) 把某一个学习模式的值作为输入层单元的输出 $\{I_i\}$, 用输入层到隐层的权值 $\{W_{ji}\}$ 和中间层单元的阈值 θ_j 求出中间层单元 j 的输出 H_j :

$$H_j = f(\sum_i W_{ji} \cdot I_i - \theta_j)$$

2) 用中间层的输出 $\{H_j\}$, 中间层到输出层的权值 $\{V_{kj}\}$ 以及输出层单元 k 的阈值 γ_k 求出输出层单元 k 的输出 Y_k :

$$Y_k = f(\sum_j V_{kj} \cdot H_j - \gamma_k)$$

3) 由学习模式的教师信号和输出层的输出得到第 m 个模式的第 k 个节点的误差 e_{mk} :

$$e_{mk} = Y_{mk} - \bar{Y}_{mk}, \text{ 而 } e = \sum_m \sum_k e_{mk} \text{ 和 } E = \sum_m \sum_k e_{mk}^2$$

显然在计算误差的过程中, 我们将阈值看成是输入为 -1 的权值;

(3) 选择继承: 评价各个权值及阈值, 对适应度 F_i 的个体赋予其选择概率 P_i :

$$P_i = F_i / \sum_{j=1}^N F_j$$

在实际训练中, 一般将适应度最大的个体直接遗传给下一代。

(4) 交叉: 只用一点交叉, 但在训练中将整个 0/1 串分为两部分: 连接权部分和阈值部分, 对于图 6.4 的网络结构, 前 48 位 (bit0~47) 为连接权子串, 而后 24 位 (bit48~71) 为阈值子串。对这两子串分别进行一点交叉, 基于这种考虑的原因是在整个训练过程中将权值与阈值分开。

(5) 变异: 以变异概率 P_m 随机地改变 0/1 串中的某些位。

(6) 从当前父代和子代的各种取值中重新排序选择出 N 个适应度较大的个体作为下一代的样本。转步骤(2)重新进行训练, 终止条件为群体适应度趋于稳定或误差 e 小于某一给定值或已到达预定的进化代数, 训练结果如表 6.2 所示。

表 6.2 训练的结果(取 $F=1/E$, $P_c=0.04$, $P_m=0.04$, $N=50$)

| | 最终权值分布 | 10次训练中的收敛次数 | 收敛速度 |
|----|---|-------------|------|
| GA | $w(1,3)=-18.407$ $w(1,4)=-9.225$ $w(2,3)=-13.123$ $w(2,4)=-10.630$ $w(3,5)=-19.380$ $w(4,5)=-19.844$ $Q(3)=4.641$ $Q(4)=-15.781$ $Q(5)=-10.938$ | 10 | 由快至慢 |
| BP | $w(1,3)=5.895$ $w(1,4)=4.184$ $w(2,3)=5.904$ $w(2,4)=4.185$ $w(3,5)=8.207$ $w(4,5)=-8.912$ $Q(3)=-2.545$ $Q(4)=-6.426$ $Q(5)=-3.816$ | 8 | 较快 |

2. 权值与结构共同进化

同样采用二进制编码方案,以图6.6为例,假设网络已定为3层,同层节点之间没有连接权相连,节点的转换函数仍为S型函数。在此前提下,我们来进化网络拓扑结构中的连接性质(即两节点间是否相连)和权值。

(1) 0/1串编码:包括粒度、连接性质和权值三部分。粒度即编码长度,连接性质决定对应的两节点间有无权值相连,权值说明两节点间的联系程度。

若初始粒度为3,则图6.6所示网络对应的0/1串可能为:

011 1 110 1 010 1 101 1 111 1 010 1 100 1 001 1 100

粒度 权值 连接性质

若初始粒度为2,另一0/1串为:

10 0 00 1 01 0 00 0 01 1 00 1 11 0 11 1 01

粒度

也可简化为:

10 0 1 01 0 0 1 00 1 11 0 1 01

请注意:若网络的两节点间没有连接,则在编码时,连接性质为“0”,而连接权可随机设置。

(2) 适应度函数:同样可选择 $F = 1/E$ 。

(3) 交叉:采用一点交叉和两点交叉,将完整的一个0/1串分为两部分:粒度部分和结构部分。交叉时分别对这两部分进行。例如,对上面两个0/1串进行一点交叉的过程为:

011 1 110 1 010 1 101 1 111 1 010 1 000 1 001 1 100

10 0 00 1 01 0 01 0 10 1 00 1 11 0 11 1 01

↓ 由于粒度不同,需对较小粒度串转换,方法是在
每一连接关系对应的权值子串后增加一个“0”或“1”

0011 1 110 1 010 1 101 1 111 1 010 | 1 100 1 001 1 100

0010 0 000 1 010 0 010 0 101 1 000 | 1 111 0 110 1 010

↓ 交叉(一点交叉)

0011 1 110 1 010 1 101 1 111 1 010 1 111 0 110 1 010

0010 0 000 1 010 0 010 0 101 1 000 1 100 1 001 1 100

简化较小粒度串,方法是删除每一连接权对应子
↓ 串的最后位“0”或“1”,若连接性质对应位为
“0”,则再删除对应连接权字符串

011 1 110 1 010 1 101 1 111 1 010 1 111 0 110 1 010

10 0 1 01 0 0 1 00 1 10 1 00 1 10

(4) 变异:分别对粒度、连接性质和权值三部分进行,赋予不同的变异概率:粒度变异概率 P_{mg} ,连接变异概率 P_{mc} 和权值变异概率 P_{mv} 。当粒度发生变化时,其后的权值串就要进行转换。

(5) 另外设计一个三元遗传选择算子 TGA:这个算子针对三个编码个体进行。

设三个个体为 x_1, x_2, x_3 , 其适应度值分别为 $F(x_1), F(x_2), F(x_3)$, 并且 $F(x_1) \geq F(x_2) \geq F(x_3)$, 另外设 TGA 的结果为 x_4 , 则 TGA 操作可描述如下:

for x_1, x_2, x_3, x_4 对应的个体中的每一位 i do


```

        if  $x_{1_i} = x_{2_i}$  then  $x_{4_i} = x_1$ 
        else  $x_{4_i} = \text{negate}(x_{3_i})$ 
    endif
end for

```

其中 $\text{negate}(x)$ 为求 x 的反。

例如设 $x_1 = 1011101$, $x_2 = 0110110$, $x_3 = 0011010$, $F(x) = x$ 则 $F(x_1) = 93$, $F(x_2) = 54$, $F(x_3) = 26$, 可得 $x_4 = 1110101$, 且 $F(x_4) = 117$ 。

(6) 终止条件: 与前面类似, 我们设定 $P_c = 0.87$, $P_{mg} = 0.03$, $P_{mc} = 0.07$, $P_{mw} = 0.08$, $N = 50$, 训练40代后, 将图6.6网络结构进化成图6.4网络结构。

6.1.4 神经网络学习规则的进化

学习规则在神经网络系统中决定了系统的功能。在以前的神经网络训练中, 学习规则都是事先设定的, 如 BP 网络用的是广义 δ 规则, 未必完全合适。我们自然想到采用遗传算法来设计神经网络中的学习规则, 使之能适应问题和环境的要求。进化学习规则的过程可描述如下^[2]:

- (1) 随机产生 N 个个体, 每个个体表示一个学习规则;
- (2) 构造一个训练集, 其中的每个元素代表一个结构和连接权是随机设定的或预先确定的神经网络, 然后对训练集中的元素分别用每一个学习规则进行训练;
- (3) 计算每个学习规则的适应度;
- (4) 根据适应度进行选择;
- (5) 对每个被编码的学习规则(个体)进行遗传操作产生下一代个体;
- (6) 重复(2)~(5), 直到达到目的为止。

1. 学习参数的进化

在学习规则中有许多参数, 这些参数用来调整网络的行为, 比

如学习率可以加快网络训练的速度。进化学习参数的方法有两种：

(1) 在编码阶段，将学习参数和结构一同编码，然后对结构进化，同时进化了学习参数。如 BP 网络中的 lrate 参数，它决定了初始权值和阈值的大小。在对网络结构编码时，可另加一参数子串，进化的过程与结构进化过程完全一致。

(2) 假设网络结构已预先确定，只对参数编码优化。

2. 学习规则的进化

这种进化的对象是学习规则本身或权值调整规则，它更能使进化后的网络适应动态环境。与进化连接权和结构不同，学习规则的进化针对的是 ENN 的动态行为。进化时的最大问题是如何将学习规则编码为字符串，其方法是：

「假设1」 学习规则对所有的连接都一致。

也就是说，所有的权值在改变时，都遵循同样的变化规则。现在的神经网络都遵循这一假设。

「假设2」 权值的改变只依赖于输入节点激活值、输出节点激活值、当前连接权值等局部信息。

「假设3」 学习规则是一线性函数。这时，学习规则可表示为

$$\Delta W(t) = \sum_{k=1}^n, \sum_{i_1, \dots, i_k=1}^n \left(\theta_{i_1}, \dots, i_k, \prod_{j=1}^k x_{i_j}(t-1) \right)$$

其中 x_1, \dots, x_n 是局部信息， t 代表当前代， $\Delta w(t)$ 即权值的变化，所有的 θ 是由进化确定的系数。这样，如果预先确定局部信息，则编码时只对所有的 θ 及比例参数进行即可。

总之，对学习规则的进化研究还刚刚开始，这是一个非常有前途的研究领域。

6.2 遗传算法与模糊集理论

模糊集理论和模糊逻辑自60年代出现以来，在人工智能、信息处理以及模糊控制等领域已获得广泛应用。其中，作为模糊推理特

征的是采用 IF... THEN 形式的模糊推理规则以及由此产生的模糊模型。本节首先讨论基于遗传算法的模糊推理规则的优化和隶属度函数的调整；然后介绍遗传算法在模糊模式识别中的应用。

6.2.1 基于遗传算法的模糊推理规则的优化

关于模糊推理规则的优化，以前主要采用神经网络方法^[9-10]。但神经网络由于存在网络规模和网络结构较为复杂以及学习收敛性问题，因此最近不少文献^[11]开始采用遗传算法研究模糊 IF - THEN 规则的优化及隶属度函数的调整。

下面介绍基于遗传算法的隶属度函数的优化和自动调整方法，包括：初始集团个体的随机产生方法，交叉、变异等遗传操作的实现，IF... THEN 规则的自动调整，各个个体的评价、淘汰和选择以及适应度函数的确定等。

具体方法是先由遗传算法产生大致的模糊模型，然后由增量规则进行调整，获得优化的模糊模型。这里所指的优化的定义是指实际计算输出与教师信号的输出误差在目标值以内收敛的个体，或者是经一定世代后，输出误差最小的个体。其中遗传操作可按如下方法进行。

1. 编码

隶属度函数有多种形式，为讨论方便，模糊推理规则的条件部以可变的三角形隶属度函数为例，结论部以实数值构成。这样，对

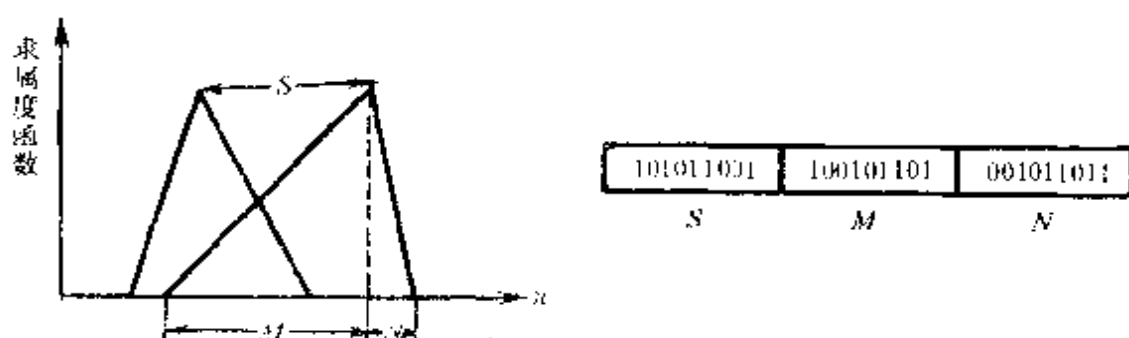


图 3-7 隶属度函数的编码

隶属度函数的每一个个体的编码如图6.7所示,由三部分组成:

- (1) 条件部隶属度函数的顶点之间的距离 S ;
- (2) 以顶点的横坐标为基点,至底边两端点的距离 M ;
- (3) 结论部的实数值 N 。

2. 适应度函数

为了对各个体进行定量评价,必须设定适应度函数作为相应的评价函数。

为此需计算教师信号的输出值和对应于输入值的计算输出值之间误差的平方和 E ,这个数值小表示适应度函数高,并定义适应度函数如下式:

$$F(S_i) = E + A \times N$$

其中, S_i 表示第 i 个个体, E 表示教师信号的输出值和计算值的误差的平方和, A 是常数, N 表示条件部隶属度函数个体的总和。

3. 交叉操作

可在某一世代群体中,随机选取 A 和 B 两个个体,进行遗传算法的交叉操作,以隶属度函数顶点之间的距离 S 的交叉操作为例。可以交叉点为界,分为左右两侧,交叉操作后,产生如下四种情况(如图6.8):

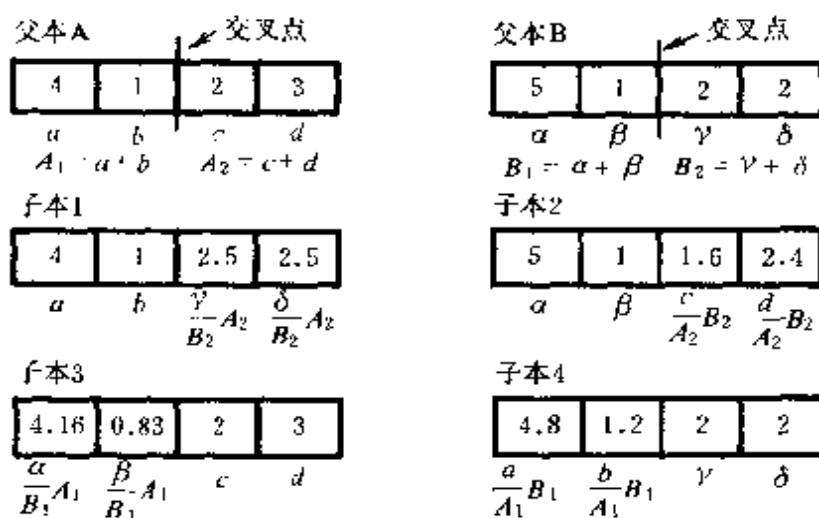


图 6.8 交叉操作

(1) 交叉点左侧, 继承 A 左侧的遗传信息; 右侧按比例继承 A 的部分信息;

(2) 交叉点左侧, 继承 B 左侧的遗传信息; 右侧按比例继承 B 的部分信息;

(3) 交叉点左侧, 按比例继承 A 部分信息; 右侧继承 A 右侧的遗传信息;

(4) 交叉点左侧, 按比例继承 B 遗传信息; 右侧继承 B 的遗传信息。

其它遗传操作如变异、选择等可按常规方法, 通过遗传算法对隶属度函数进行初调, 确定隶属度函数的粗略形状, 然后再采用如下方法进行细调。

4. 隶属度函数的细调整

下面介绍采用 δ 规则对条件部三角形隶属度函数的底边以及结论部的实数值进行调整的方法。

(1) 三角形隶属度函数底边的调整

首先, 以下式表示第 K 个教师模式的计算输出值 Y_K :

$$\mu_i = \prod_{j=1}^m A_{ij}(x_j)$$
$$Y_K = \frac{\sum_{i=1}^n \mu_i W_i}{\sum_{i=1}^n \mu_i}$$

其中: x_j 为第 j 个输入, m 为输入的个数, n 为模糊规则数, μ_i 是推理规则 i 的适应度, A_{ij} 为模糊变量, W_i 为后部件的实数值。则前部件的底边的调整式如下所示:

$$E = \sum_{K=1}^p \frac{1}{2} (Y_K - Y_{(K)})^2$$

$$D_{(i+1)} = D_{(i)} - \alpha \frac{\partial E}{\partial D_{(i)}}$$

$$= W_{(i)} - \alpha \frac{\prod_{j=1}^m A_{ij}(x_j)}{\sum_{i=1}^n \mu_i} \times (Y_K - Y_{iK})(W_i - Y_K)$$

其中: E 为第 K 个教师信号的输出值 Y_{iK} 和计算值 Y_K 的误差平方和, $D(t)$ 为第 t 回学习中, 隶属度函数底边端点的距离, α 为学习参数。

(2) 结论部的调整方法

后部件实数值 W_i 的调整可按如下公式:

$$W_i(t+1) = W_i(t) - \beta \frac{\partial E}{\partial W_i} = W_i(t) - \beta \frac{\mu_i}{\sum_{i=1}^n \mu_i} (Y_K - Y_{iK})$$

其中, β 为学习参数。

以上是对模糊规则的条件部和结论部进行优化调整的方法。

6.2.2 遗传算法在模糊模式识别中的应用

基于模糊 IF-THEN 规则的模糊识别的研究可采用图 6.9 所示的方法。

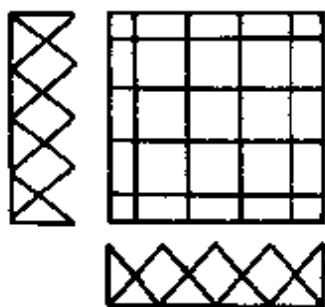


图 6.9 模糊分割

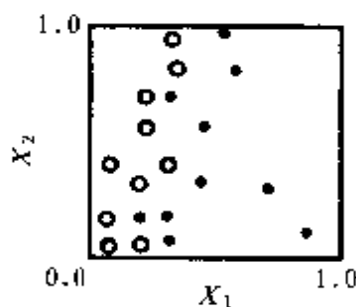


图 6.10 模式识别问题举例

由模糊方格将模式空间进行模糊分割, 对于各模糊部分空间生成模糊 IF-THEN 规则; 这时模糊分割方法对模糊识别系统的性能有很大的影响。例如, 模糊分割很细的情况对学习用的识别能力高, 但对评价用的模式性能不好; 相反, 采用粗的模糊分割的情况, 对

复杂的模式识别效果不好。因此,要得到良好性能的模式识别效果,必须有适当的模糊分割。

例如,图 6.10 所示黑白两种圆的模式识别问题,从图 6.10 直观的观察,左半边需要细的模糊分割,对于右半边需要粗的模糊分割,因此如图 6.9 采用单一的模糊格子的情况要进行合适的模糊分割是很难的。对此问题,文献^[13]提出一种同时采用多种模糊分割的称为分散型模糊规则概念的模糊识别系统。图 6.12 表示对应于六种不同的模糊分割的模糊 IF — THEN 规则,同时进行模糊推理。这种方法是同时采用粗的模糊分割和细的模糊分割以解决模糊分割的困难问题。但是,采用多种模糊分割,将引起模糊 IF — THEN 规则数大大增加,这又是我们所不希望的。

下面介绍一种从对应于多种模糊分割的多个模糊 IF — THEN 规则中选出最必须的最小限度规则,并建立小型高性能模糊识别系统的方法。

为清楚明了起见,先介绍一种基于一般模糊 IF — THEN 规则的模式识别。它是首先将构成模糊识别系统的问题作为一种选择模糊 IF — THEN 规则的组合最优化问题。该问题的目标函数是正确率最大化和规则数最小化;其次,介绍采用遗传算法,将模糊 IF — THEN 规则的集合作为一个个体,将上述两个目标函数的加权和定义为个体的适应度;然后由遗传操作进行优化组合。具体方法如下:

1. 由模糊 IF — THEN 规则的模式识别

在讨论由遗传算法选择规则之前,先举例说明模糊 IF — THEN 规则的生成方法。为了说明简单,模式空间设定为 $[0,1] \times [0,1]$ 的二维平面,学习用的数据被分成 M 个群 (C_1, C_2, \dots, C_M) 的 M 个模式 $x_p = (x_{p1}, x_{p2}), p = 1, 2, \dots, M$ 。二维模式空间的各维由模糊分割成 K 个模糊集合 $\{A_1^K, A_2^K, \dots, A_K^K\}$, 基于模糊 IF — THEN 规则的模糊识别系统可采用如下规则:

$$\begin{aligned} \text{Rule } R_i^K: & \text{If } x_1 \text{ is } A_i^K \text{ and } x_2 \text{ is } A_j^K \\ & \text{Then } (x_1, x_2) \text{ belongs to } C_0^K \end{aligned} \quad (6.1)$$

With $CF = CF_j^K$

其中, R_j^K 是规则的标志; A_i^K 和 A_j^K 是规则条件部模糊集合; C_j^K 是规则的结论; CF_j^K 是规则的可信度。

对于 $K=2$ 和 $K=3$ 的情况, 模式空间各维的模糊分割和模糊集合的标志关系如图 6.11 所示。其中 S_i 表示规则的顺序符号, 它将在后面遗传算法中应用。

作为条件部的模糊集合, 图 6.11 采用三角形模糊集合。当然, 也可采用其它形状。对三角形模糊集合 A_i^K 的隶属度函数写成 $\mu_i^K(\cdot)$:

$\mu_i^K(x) = \max\{1 - |x - a_i^K|/b^K, 0\}, i=1, 2, \dots, K$, 其中, a_i^K 是三角形模糊集合的中心; b^K 是从中心至两边的宽度; $a_i^K = (i-1)/(K-1), i=1, 2, \dots, K; b^K = 1/(K-1)$ 。

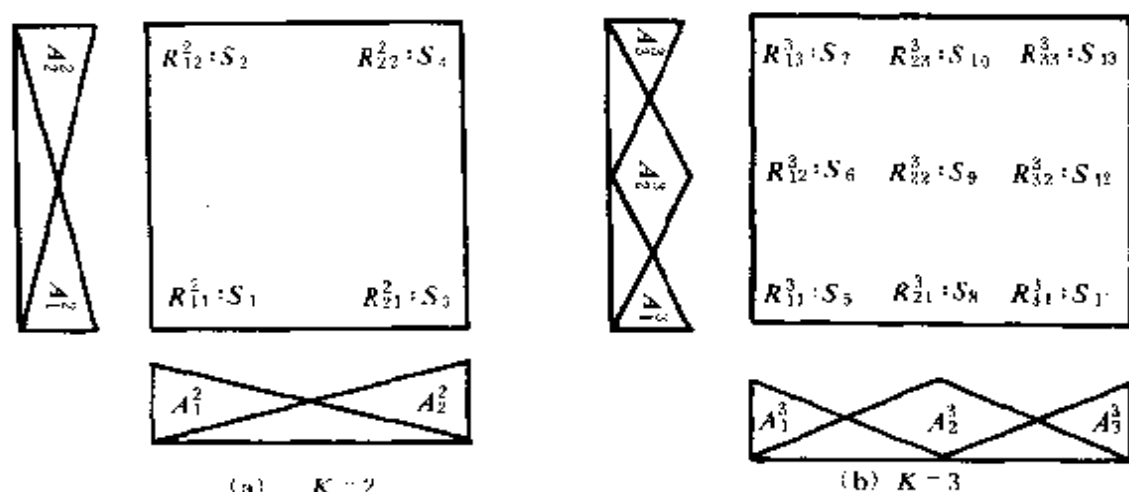


图 6.11 模糊 IF-THEN 规则 R_j^K 及标志

式(6.1)所示 IF-THEN 规则的结论 C_j^K 及可信度 CF_j^K , 可由学习用模式 $x_p (p=1, 2, \dots, M)$ 按如下程序自动生成:

第1步: 对第 t 群 (C_t), 按如下定义计算 $\beta_{C_t} (t=1, 2, \dots, M)$:

$$\beta_{C_t} = \sum_{p \in C_t} \mu_i^K(x_{p_1}) \cdot \mu_j^K(x_{p_2})$$

第2步: 求 β_{C_t} 最大的群 CX , 即求满足下式的群 $CX: \beta_{C_t} = \max\{\beta_{C_1}, \beta_{C_2}, \dots, \beta_{C_M}\}$, 则模糊 IF-THEN 规则的结论 C_j^K 就为 CX , 此

时,如果有两个以上的群取最大值的情况,则模糊 IF — THEN 规则的结论 C_j^K 就不能唯一确定。为方便,可令 $C_j^K = \emptyset$ 。

第3步:模糊 IF — THEN 规则的结论 C_j^K 为 \emptyset 时,可信度 $CF_j^K = 0$, C_j^K 不为 \emptyset 的情况,模糊 IF — THEN 规则的可信度 CF_j^K 按下式求之:

$$CF_j^K = |\beta_{C_j} - \beta| / \sum_{i=1}^M \beta_{C_i}$$

$$\beta = \sum_{i=1, C_i \neq C_j}^M \beta_{C_i} / (M - 1)$$

对于结论部为 \emptyset 的规则,由下面将要介绍的遗传算法产生新规则。

对于图 6.10 的例题,设定 $K=2 \sim 7$ 情况生成的模糊 IF — THEN 规则如图 6.12 所示,图中各区域所对应模糊部分空间所生成的模糊 IF — THEN 规则如下:

斜线区域:结论 C_j^K 是第一群(黑圆)的规则;

点的区域:结论 C_j^K 是第二群(白圆)的规则;

空白区域:结论 C_j^K 是 \emptyset 的虚规则。

由图 6.12 可以看出:模糊分割较细的情况(K 值大),结论为 \emptyset 的空规则数较多。

按上述程序生成的模糊 IF — THEN 规则集合的全部集合 S_{ALL} 定义为:

$$S_{ALL} = \{R_j^K | i = 1, 2, \dots, K; j = 1, 2, \dots, K; K = 2, 3, \dots, L\}$$

其中, L 表示各维模糊分割数的最大值(整数);另外,用 $S(S \subseteq S_{ALL})$ 表示 S_{ALL} 中用于模糊识别系统的部分规则集。这时,未知模式

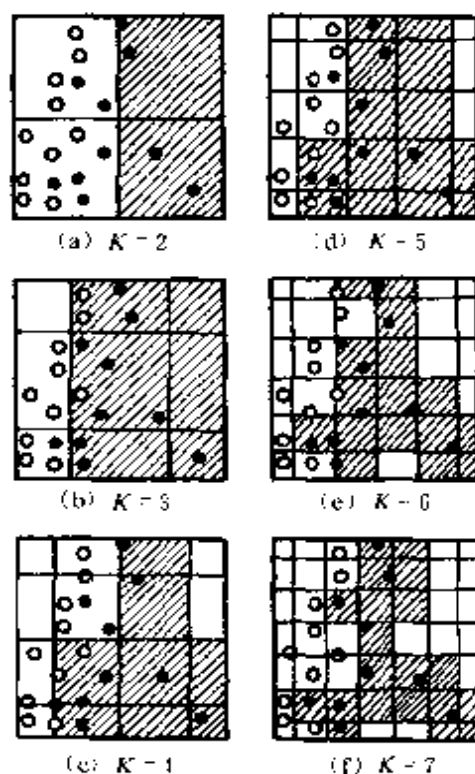


图 6.12 生成的模糊规则

$x_p = (x_{p1}, x_{p2})$ 的识别可按如下方法进行:

第1步: 对于第 t 群(C_t), 按如下定义计算 $ac_t (t=1, 2, \dots, M)$:

$$ac_t = \max \{ \mu_i^K(x_{pi}) \cdot \mu_j^K(x_{pj}) \cdot CF_{ij}^K | C_{ij}^K = C_t; R_{ij}^K \in S \}$$

第2步: 由 ac_t 中求最大的群 CX, 即求满足下式的群 CX:

$$a_{C_x} = \max \{ a_{C_1}, a_{C_2}, \dots, a_{C_M} \}$$

其中, 如果有两个以上的群都有最大值的情况, 则未知模式 x_p 就很难识别。

最基本的模式识别系统是具有相应的某一 K 值的模糊 IF — THEN 规则集:

$$S = \{ R_{ij}^K | i = 1, 2, \dots, K; j = 1, 2, \dots, K \}$$

对应于图 6.13 所示识别结果, $K=2 \sim 7$, 即图 6.13 是图 6.12 中所示的模糊分割及其相应模糊 IF — THEN 规则识别结果。其中, 涂黑区域表示不能识别的区域, 在这些区域, 对未知模式不能识别。

由图 6.13 可知, 模糊分割越粗(K 值小的情况), 学习用的模式存在误识别; 模糊分割细(K 值大的情况), 存在不能识别的区域。因此, 由模糊分割所得的模糊 IF — THEN 规则所构成的模糊识别系统要设定合适的 K 值是十分困难的。

2. 组合最优化问题的定式化

上面介绍由学习模式生成模糊 IF — THEN 规则的方法, 下面介绍由所生成的全部模糊 IF — THEN 规则集 S_{ALL} 选出所需合适的规则集 $S (S \subseteq S_{ALL})$ 的问题, 这是一个组合最优化问题。

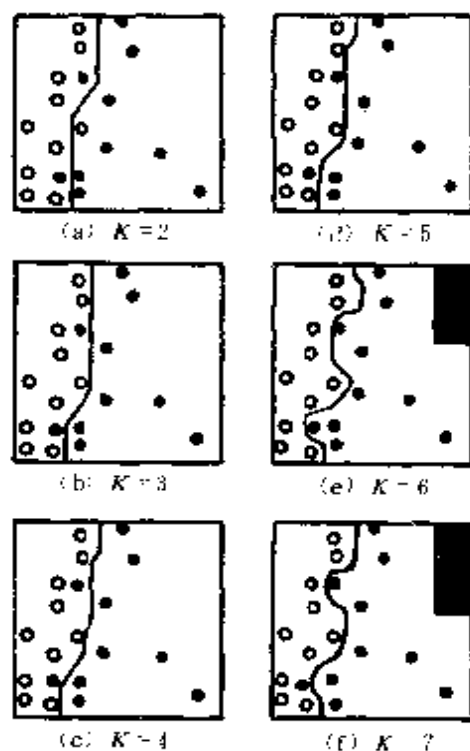


图 6.13 识别结果

定义由规则集 S 能正确识别学习用模式的数目为 $NCP(S)$ 。另外, 规则集 S 中所含模糊 IF — THEN 规则数为 $|S|$ 。则为建立小型高性能模糊识别系统为目的, 其目标函数可考虑为 $NCP(S)$ 最大和 $|S|$ 最小化, 亦即为如下两目标的组合最优化问题:

目标函数: 最大化 $NCP(S)$

最小化 $|S|$

约束条件: $S(S \subseteq S_{ALL})$

对这个问题, 相当于从规则集 S 中选出所需的模糊 IF — THEN 规则的问题, 并希望能够用尽量少的规则来正确识别尽量多的模式。

如果全部规则集 S_{ALL} 所含规则数为 N , 则此问题的实际可能解的总数为 2^N , 其计算量很大, 因此必须利用组合优化方法来求解, 并探索求得优化解的方法。为此必须采用下节的遗传算法。

3. 采用遗传算法的最优化方法

(1) 适应度函数的定义。

上面所定义的两目标组合最优化问题采用遗传算法是很合适的。在此, 当权值用 W_{NCP} 和 W_S ($W_{NCP}, W_S > 0$) 时, 则上述两目标函数可表示为:

目标函数: 最大化 $W_{NCP} \cdot NCP(S) - W_S \cdot |S|$

采用此目标函数, 则可定义规则集 S 的适应度 $f(S)$ 为:

$$f(S) = \max\{W_{NCP} \cdot NCP(S) - W_S \cdot |S|, 0\}$$

(2) 个体的定义。

用遗传算法求解组合最优化问题, 首先必须解决如何表现各种可能解的符号序列的个体。兹介绍一种表现个体的方法, 即把规则集 S 表示成长度为 N ($N = |S_{ALL}|$) 的序列:

$$S = S_1, S_2, \dots, S_N$$

其中, $S_r = 1, 0$ 或 -1 , $r = 1, 2, \dots, N$; $S_r = 1$ 表示第 r 号规则是 S 中的规则; $S_r = 0$ 表示第 r 号规则是虚规则; $S_r = -1$ 表示第 r 号规则不是 S 中的规则。

各个规则例如图 6.12 所示, 按 S_1, S_2, \dots, S_N 顺序排列。

(3) 遗传操作。

本节所采用遗传算法由如下基本操作组成：

1) 第一代个体群的组成。

按以上规定，对虚规则以 $S_r=0$ 表示；对虚规则以外的模糊 IF-THEN 规则，随机以 $S_r=1$ 或 $S_r=-1$ 表示。并由此操作，生成第一代个体群。

2) 确定选择概率。

以 ϕ_t 表示第 t 代个体的集合。这时，由交叉操作产生下一代个体群，并按如下选择概率 $P(S)$ 公式选择个体 S ：

$$P(S) = f(S) / \sum_{S' \in \phi(t)} f(S')$$

3) 交叉操作。

按上述选择概率选出两个个体进行交叉操作，生成两个新的个体，并按此方法不断重复进行，生成下一代个体群。

4) 变异操作。

对由交叉操作生成的各个体 S_r ，按变异概率进行如下变异操作，即

$$S_r \rightarrow S_r \cdot (-1)$$

即由 $1 \rightarrow -1$ 或由 $-1 \rightarrow 1$ ，但对 $S_r=0$ 的虚规则， S_r 值不变。

5) 优化保存策略。

对于前一代中最佳的个体，即适应度 $f(S)$ 最大的个体，必须在下一代中保存下来。通过这一操作，至第 t 代生成的最佳的个体，必须在第 $t+1$ 中。

4. 实验数据

对于图 6.10 采用遗传算法进行模式识别时，各世代的个体数为 10；总世代数（运算停止基准）为 1000；变异概率 $P_m=0.01$ ；权值为： $W_{NCP}=10, W_S=1$ ；规则集 $S_{ALL}=\{R_{ij}^k | i=1,2,\dots,K; j=1,2,\dots,K, K=2,3,4,5,6\}$ 。

如图 6.12 (a)~(e) 所示，利用遗传算法，从对应于 $K=2\sim6$

的模糊 IF — THEN 规则集中选出规则集 S ，确定能够正确识别的最大 K 值。例如在图 6.13 中，当 $K=6$ 时，能够正确进行模式识别。按原来分散型模糊规则集的方法，规则集总数 $|S_{ALL}|=90$ ，规则集 S 的总数 $S=20^{96}\approx 1.2\times 10^{27}$ ；采用遗传算法，规则集 S 的总数为 $10\text{个个体}\times 1000\text{代}=10000$ 。

由此设定的数值实验，并得到能正确进行模式识别的规则集 S ，由各数值实验所得规则集 S 中需含最大模糊 IF — THEN 规则数为 8，最小为 5，平均约为 6.5。

该数值实验所得到规则集 S 中所含模糊 IF — THEN 规则以及对应的识别结果如图 6.14 所示。

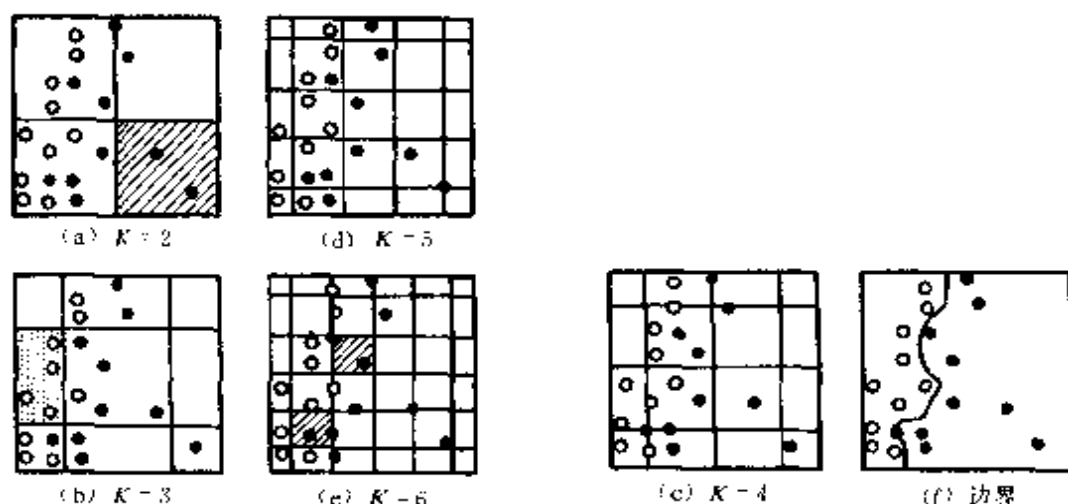


图 6.14 采用遗传算法的识别结果

图 6.14(a)~(e)划出由选出的 5 个模糊 IF — THEN 规则以及对应的由斜线及点所表示的区域，图 6.14(f)为其识别结果，全部学习用模式都正确识别。另外，由图 6.14(f)可看出，图 6.13(c)以及图 6.13(f)的涂黑区域不存在，即没有不能识别的领域，即由图 6.14 的 5 个模糊 IF — THEN 规则能将模式空间完全覆盖。

将图 6.14 所示结果与图 6.12 及图 6.13 比较，可看出遗传算法的有效性，亦即用很少数量的模糊 IF — THEN 规则得到良好的识别结果。

在上述遗传操作过程中，可通过调整遗传操作的各种参数的设

定值，来改善遗传算法的性能和效果。例如可选择遗传操作的如下不同参数设定值：

各世代的个体数：5，10，50；

变异概率： $P_m=0.1, 0.01, 0.001$ 。

其它设定参数如总代数、权值及全部规则集合 S_{ALL} 等不变。

当采用保存“优化”策略时，所得的数值实验结果如表6.3和表6.4。可以看出变异概率对规则集所含的规则数有较大的影响。

表 6.3 **规则集 S 所含规则数**

| 变 异 概 率 | 每 代 个 体 数 | | |
|---------|-----------|------|------|
| | 5 | 10 | 50 |
| 0.1 | 15.0 | 16.1 | 15.6 |
| 0.01 | 6.1 | 6.5 | 7.4 |
| 0.001 | 8.2 | 6.6 | 6.0 |

表 6.4 **正确识别模式数目**

| 变 异 概 率 | 每 代 个 体 数 | | |
|---------|-----------|------|------|
| | 5 | 10 | 50 |
| 0.1 | 20.0 | 20.0 | 20.0 |
| 0.01 | 19.9 | 20.0 | 20.0 |
| 0.001 | 19.7 | 20.0 | 20.0 |

表 6.5 **规则集所含规则数**

| 变 异 概 率 | 每 代 个 体 数 | | |
|---------|-----------|------|------|
| | 5 | 10 | 50 |
| 0.1 | 25.9 | 24.6 | 23.0 |
| 0.01 | 28.5 | 26.0 | 20.5 |
| 0.001 | 34.2 | 29.3 | 23.1 |

表 6.6

正确识别模式数目

| 变异概率 | 每代个体数 | | |
|-------|-------|------|------|
| | 5 | 10 | 50 |
| 0.1 | 20.0 | 20.0 | 20.0 |
| 0.01 | 20.0 | 20.0 | 20.0 |
| 0.001 | 19.5 | 19.7 | 19.6 |

表 6.5 和表 6.6 为不采用保存“优化”策略的情况。可看出采用保存“优化”策略将使遗传算法的性能提高。

6.3 进化算法^[16]

6.3.1 引言

从本世纪40年代起,生物模拟就构成了计算科学的一个组成部分,像早期的自动机理论,就是假设机器是由类似于神经元的基本元素组成的,它向人们展示了第一个自复制机模型。这些年来诸如机器能否思维、基于规则的专家系统是否能胜任人类的工作,以及神经网络能否使机器具有看和听的功能等有关生物类比的问题已成为人工智能关注的焦点。最近生物计算在机器昆虫和种群动态系统模拟上所取得的成功激励越来越多的人致力于人工生命领域的研究。当今,计算机科学家和分子生物学家已开始携手进行合作研究,生物类比也得到了更为广泛的应用。

自然界生物体通过自身的演化就能使问题得到完善的解决。这种才能让最好的计算机程序也相形见绌。计算机科学家为了某个算法可能要耗费数月甚至几年的努力,而生物体则是通过进化和自然选择这种非定向机制来达到目的。

近30年的不断的研究和应用已经清楚地表明了模拟自然进化的搜索过程可以产生非常鲁棒(robust)的计算机算法,虽然这些模型还只是自然界生物体的粗糙简化。进化算法就是基于这种思想发展

起来的一类随机搜索技术，它们是模拟由个体组成的群体的集体学习过程。其中每个个体表示给定问题搜索空间中的一点。进化算法从任一初始的群体出发，通过随机选择（在某些算法中是确定的）、变异和交叉（在某些算法中被完全省去）过程，使群体进化到搜索空间中越来越好的区域。选择过程使群体中适应性好的个体比适应性差的个体有更多的复制机会，交叉算子将父代信息结合在一起并将他们传到子代个体，变异在群体中引入了新的变种。

在计算机科学中，进化实质上是一种优化处理过程；但这种过程与传统的优化方法不同，传统的优化方法都是用代价函数来衡量动作的行为从而通过选择一个好的动作使操作的对象得到优化；绝大多数典型的优化方法是通过计算代价函数的梯度值或高阶统计值进行优化的，一般情况下，这类方法只能得到局部极优值，并且容易受到随机干扰的影响。而进化的方法符合达尔文“适者生存”和随机信息交换思想，既消除解中不适应因素，又利用了原有解中的知识，从而使优化过程加快，最终获得全局极优解。

进化算法(Evolutionary Algorithms, 简称 EA)具有如下特点：

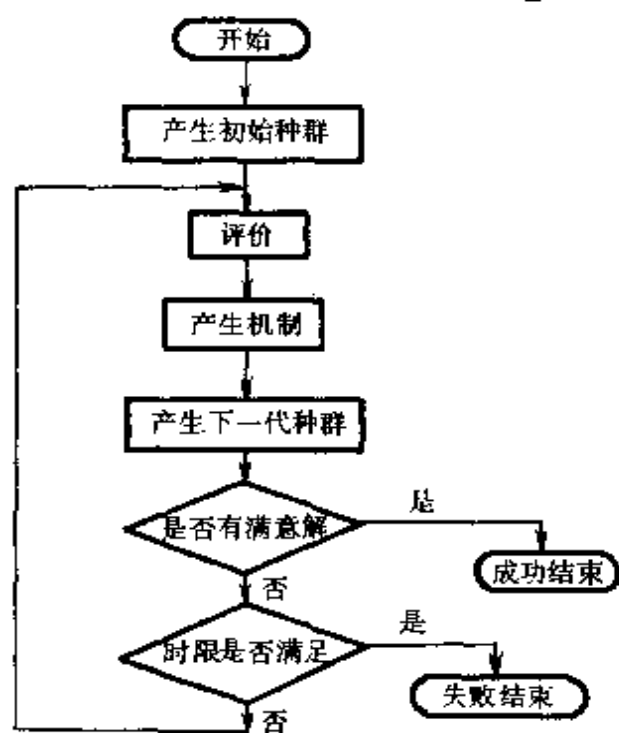


图 5.10 进化算法的过程

(1) 个体(候选解)是选择操作的主要目标；

(2) 变异操作是一种偶然现象，即变异以一很小的概率进行；随机处理在进化中起着主要作用；

(3) 种类的改变主要由交叉和变异得到；

(4) 新种是从旧种开始，经历突变、自然选择和隔离等过程的渐次分化得到；种类的延续不是完全连续的；

- (5) 不是所有种类的改变都是自然选择的结果;
- (6) 种类的进化是适应其环境的结果, 并且是多种多样的;
- (7) 在进化的过程中, 具体选择哪一个个体或种类不是确定性的。

进化算法的大致过程可用图6.15描述。

其中, 初始种群是针对具体问题随机设定的; 评价操作是根据预先定义的一个评价函数来计算当前种群(当前代)中各个个体(候选解)在环境中的“适应度”(fitness); 产生机制的功能是通过选取一些进化方法(如选择、交叉和变异)从当前种群中产生出新的个体; 接着的操作就是从当前种群和新产生的个体中得到下一代种群; 这就完成了一次进化; 最后, 算法的终止条件是当前种群中存在某个候选解满足要求或允许进化的时限已到。在这个过程中, 选取不同的进化方法决定了进化算法的不同种类。

目前研究的进化算法主要有三种典型的算法: 遗传算法、进化规划(EP, Evolutionary Programming)和进化策略(ES, Evolution Strategies)。尽管它们之间很相似, 但历史上这三种算法是彼此独立发展起来的, 遗传算法由美国 J. Holland 创建^[12], 后由 K. De Jong, J. Grefenstette, D. Goldberg 和 L. Davis 等人进行了改进; 进化规划最早由美国的 L. J. Fogel, A. J. Owens 和 M. J. Walsh 提出^[13], 最近又由 D. B. Fogel 进行了完善; 进化策略是由德国的 I. Rechenberg 和 H. P. Schwefel 建立的^[14]。

群体搜索策略和群体中个体之间的信息交换是进化算法的两大特点。它们的优越性主要表现在: 首先, 进化算法在搜索过程中不容易陷入局部最优, 即使在所定义的适应度函数是不连续的, 非规则的或有噪声的情况下, 它们也能以很大的概率找到全局最优解; 其次, 由于它们固有的并行性, 进化算法非常适合于巨量并行机; 再者, 进化算法采用自然进化机制来表现复杂的现象, 能够快速可靠地解决非常困难的问题; 此外, 由于它们容易界入到已有的模型中并且具有可扩展性, 以及易于同别的技术混和等因素, 进化算法

目前已经在最优化、机器学习和并行处理等领域得到了越来越广泛的应用^[15]。1993年德国 Dortmund 大学的 Back 等人就曾在一份研究报告中搜集了260篇有关进化算法的应用的科技文献。近年来,这一算法在各方面的应用更是发展迅速。

下面首先给出三种进化方法的统一的描述框架,然后在此统一的框架下对三种类型的进化算法进行了描述和比较,最后对进化算法中交叉和变异的相对重要性问题进行了讨论^[16]。

6.3.2 进化算法的总框架

下面首先给出一些符号表示。 $f:R^n \rightarrow R$ 记为被优化的目标函数,不失一般性,这里考虑函数最小化问题;适应度函数为 $\Phi:I \rightarrow R$, 其中 I 是个体的空间,一般不要求个体的适应值与目标函数值相等,而 f 是 Φ 的变量, $a \in I$ 记为个体, $x \in R^n$ 为目标变量, $\mu \geq 1$ 记为父代群体规模; $\lambda \geq 1$ 为子代群体规模,即在每一代交叉和变异产生的个体数;在进化代 t , 群体 $P(t) = \{a_1(t), \dots, a_\mu(t)\}$ 由个体 $a_i(t) \in I$ 组成; $r:I^\mu \rightarrow I^\lambda$ 记为交叉算子,其控制参数集为 Θ_r ; $m:I^\lambda \rightarrow I^\lambda$ 记为变异算子,其控制参数集为 Θ_m ;这里 r, m 均指宏算子,即把群体变换为群体,把相应作用在个体上的算子分别记为 r' 和 m' ;选择算子 $S:(I^\lambda \cup I^\mu) \rightarrow I^\mu$ 用于产生下一代父代群体,其控制参数集为 Θ_s ; $\Lambda:I^\mu \rightarrow \{T, F\}$ 记为停止准则,其中 T 表示真, F 表假。

有了上面的符号表示,可以把一个进化算法形式描述为:

$t = 0$

初始化 $P(0) = \{a_1(0), \dots, a_\mu(0)\}$

度量 $P(0) := \{\Phi(a_1(0)), \dots, \Phi(a_\mu(0))\}$

while $(\Lambda(P(t)) \neq T)$ do

交叉: $P'(t) = r_{\Theta_r}(P(t))$

变异: $P''(t) = m_{\Theta_m}(P'(t))$

度量: $P''(t) = \{\Phi(a''_1(t)), \dots, \Phi(a''_\lambda(t))\}$

选择: $P(t+1) = (P''(t) \cup Q)$

$t = t + 1$

end

这里 $Q \in \{\emptyset, P(t)\}$ 是选择步中附加所考虑的个体集合。在这个进化算法的统一框架下,我们将分别论述遗传算法、进化规划和进化策略。给出每种算法的一个标准形式。关于进化算法的其它修改形式,读者可参考有关文献,我们这里就不加讨论了。

6.3.3 遗传算法

本书上述各章所介绍的遗传算法可以说是最广为人知的进化算法。自从 Holland 提出遗传算法之后,De Jong 首先将遗传算法应用于函数优化,为这一应用技术奠定了基础^[17]。目前,人们对初期的遗传算法,或称为标准遗传算法进行了大量的改进,使遗传算法应用于更广泛的领域。不过这些改进算法中有许多与我们下面将要讨论的标准遗传算法有很大的差别,以至使它们与别的进化算法的界限变得模糊不清。非标准的遗传算法主要有 D. Whitley 的 Genitor 系统、J. Grefenstette 的 Samuel 系统、L. Davis 的遗传算法、Z. Michalewicz 的进化程序、Koza 的遗传编程和 L. Eshelman 的 CHC 算法。

1. 表示法和适应值度量

标准遗传算法作用于确定长度的二进制位串上,即 $I = \{0, 1\}^l$ 。对于伪布尔目标函数,这种表示法可以直接采用,为了解决函数优化的问题 $f: \prod_{i=1}^n [u_i, v_i] \rightarrow R (u_i < v_i)$ 一般是将位串分为 n 段,每段长度为 l_x , 即 $l = n \cdot l_x$, 每段表示分量 $x_i \in [u_i, v_i]$ 的二进制代码。位段译码函数 $\Gamma^i: \{0, 1\}^{l_x} \rightarrow [u_i, v_i]$ 的常见形式为:

$$\Gamma^i(a_{i1}, \dots, a_{il_x}) = u_i + \frac{v_i - u_i}{2^{l_x} - 1} \left(\sum_{j=1}^{l_x} a_{ij} 2^{j-1} \right)$$

其中 a_{i1}, \dots, a_{il_x} 记为个体 $a = (a_{11}, \dots, a_{nl_x}) \in I^l$ 的第 i 段,把位段译

码函数 Γ^i 组合成一个个体的译码函数 $\Gamma = \Gamma^1 \times \cdots \times \Gamma^m$, 则适应度函数可设置为 $\Phi(a) = \delta(f(\Gamma(a)))$, 其中 δ 为比例变换函数, 作用之一是确保适应值为正值并且最好个体的适应值最大, 常见的比例变换有线性比例、幂比例和指数比例。还有一点需要强调的是, 上面的表示法是为了使用标准遗传算法求解连续优化问题而设计的一种特殊表示法, 基于二进制代码的其他表示形式可使标准遗传算法应用到许多不同的问题领域。

2. 变异

在标准遗传算法中, 变异一般被看作为辅助算子, 它作用在位串上, 以较小的概率 P_m 随机地改变串上的每一位(即相应位上的0变为1, 或是1变为0)。变异概率 P_m 的值一般取为0.001到0.01之间, 它不依赖目标变量的维数和位串的总长。个体 (S_1, S_2, \dots, S_l) 经变异算子 $m_{(P_m)}$ 作用后变为 $(S'_1, S'_2, \dots, S'_l)$, 其中:

$$S'_i = \begin{cases} S_i & \theta_i > P_m \\ 1 - S_i & \theta_i \leq P_m \end{cases} \quad \forall i \in \{1, 2, \dots, l\}$$

这里 θ_i 是0与1之间的均匀随机数, 对串上的每一位都要重新采样。

3. 交叉

在标准遗传算法中, 交叉算子是主要的遗传算子, 它把两个不同个体上的有用段组合在一起, 交叉 $r': I^m \rightarrow I^k$ 也是作用在位串上, 以概率 P_c 对两个个体进行交叉的作用范围一般是 $[0, 6, 1, 0]$ 。两个父代个体 $S = (S_1, \dots, S_l), V = (V_1, \dots, V_l)$ 被随机地从群体中选择进行交叉, 产生的两个子代个体为:

$$S' = (S_1, \dots, S_{h-1}, S_h, V_{h+1}, \dots, V_l)$$

$$V' = (V_1, \dots, V_{h-1}, V_h, S_{h+1}, \dots, S_l)$$

其中交叉点为1到 l 之间的均匀随机整数。这种交叉算子称为一点交叉算子, 通过选择更多的交叉点并交替地交换交叉点之间的位段可以把一点交叉算子扩展到 m 点交叉算子。另外还有一些其他形式的交叉算子, 如均匀交叉算子, 它把两个父代位串上的每一位都执行一个随机判定来决定是否交换信息。事实上, 目前既没有明确的理论

也没有可靠的试验证据来决定哪一种交叉是最适当的,虽然已有一些试图解决这一问题的研究成果。

4. 选择

标准遗传算法采用的是一种依据概率选择的选择算子 $S: I^\mu \rightarrow I^\mu$, 个体 a_i 的选择概率由它的相对适应值给出

$$P_i(a_i) = \frac{\Phi(a_i)}{\sum_{j=1}^{\mu} \Phi(a_j)}, \quad \forall i \in \{1, 2, \dots, \mu\}$$

按照这个概率分布选取 μ 个个体产生下一代父代群体。显然这个选择算子对出现负适应值情形或最小化任务不适用,此时要采用适应值比例变换。

综上所述,一个标准的遗传算法可以描述为:

$t = 0$

初始化 $P(0) = \{a_1(0), \dots, a_\mu(0)\} \in I^\mu$, 其中 $I = \{0, 1\}^l$

度量 $P(0): \{\Phi(a_1(0)), \dots, \Phi(a_\mu(0))\}$, 其中 $\Phi(a_K(0)) = \delta(f(\Gamma(a_K(0))))$

while $(\wedge (P(t) \neq T))$ do

交叉: $a'_K(t) = r'_{\{P_t\}}(P(t)), \forall K \in \{1, 2, \dots, \mu\}$

变异: $a''_K(t) = m'_{\{P_m\}}(a'_K(t)), \forall K \in \{1, 2, \dots, \mu\}$

度量: $P''(t) = \{a'_1(t), \dots, a'_\mu(t): \{\Phi(a'_1(t)), \dots, \Phi(a'_\mu(t))\}$

$\Phi(a''_K(t)) = \delta(f(\Gamma(a''_K(t))))$

选择: $P(t+1) = S(P''(t))$

$t = t + 1$

end

6.3.4 进化规划

60年代中期, L. J. Fogel 等人为有限状态机的演化提出了进化规划来求解预测问题。这些机器的状态变换表是通过在对应的离散、

有界集上进行均匀随机变异来修改。进化规划根据被正确预测的符号数来度量适应值。通过变异，父代群体中的每个机器产生一个子代，父代和子代中最好的那一半被选择生存下来。基于正态分布变异，D. B. Fogel 将进化规划也被扩展到解实值问题。

1. 表示法和适应值度量

进化规划先是假设一个有界子空间 $\prod_{i=1}^n [u_i, v_i] \subset R^n$ ，其中 $u_i < v_i$ 。之后搜索区域被扩展到 $I = R^n$ ，即个体为目标变量向量， $a = x \in I$ ，进化规划把目标函数值通过比例变换到正值，同时加入某个随机改变 θ 来得到适应值 $\Phi(a) = \delta(f(x), \theta)$ ，其中 δ 是比例函数。

2. 变异

标准进化规划采用的是高斯变异算子，它把个体 x 的每个分量 x_i 作用一个标准偏差，标准偏差值取为适应值 $\Phi(a) = \Phi(x)$ 的一个线性变换的平方根，即 $m'(x) = x'$ ，其中

$$x'_i = x_i + \sigma_i \cdot N_i(0, 1),$$

$$\sigma_i = \sqrt{\beta_i \cdot \Phi(x) - r_i}, \quad \forall i \in \{1, 2, \dots, n\}$$

其中 $N_i(0, 1)$ 表示对每个下标 i 都重新采样且具有期望值为0、标准偏差为1的正态分布随机变量，系数 β_i, r_i 是特定参数，一般将 β_i 和 r_i 置为1和0，此时 $x'_i = x_i + \sqrt{\Phi(x)} \cdot N_i(0, 1)$ 。与进化策略和遗传算法不同的是在进化规划中完全没有用到交叉算子。

3. 选择

在 μ 个父代个体每个经过一次变异产生 μ 个子代后，进化规划利用一种随机 q 竞争选择方法从父代和子代的集合中选择 μ 个个体，其中 $q \geq 1$ 是选择算法的参数，具体作用过程如下：对每个个体 $a_k \in P(t) \cup P'(t)$ ，其中 $P'(t)$ 是变异后的群体，从 $P(t) \cup P'(t)$ 中随机选取 q 个个体，把它们按适应值与 a_k 进行比较，计算出其中比 a_k 差的个体数目 W_k ，并把 W_k 作为 a_k 的得分， $W_k \in \{0, \dots, q\}$ ；在所有 2μ 个个体都经过了比较过程后，按得分 $W_i (i \in \{1, 2, \dots,$

$2\mu\}$ 下降的顺序对个体排序;选择 μ 个具有最高得分 W_i 的个体作为下一代群体,更形式化地,我们有

$$W_i = \sum_{j=1}^q \begin{cases} 1, & \text{if } \Phi(a_i) \leq \Phi(a_{h_j}) \\ 0, & \text{if } \Phi(a_i) > \Phi(a_{h_j}) \end{cases}$$

$h_i \in \{1, \dots, 2\mu\}$ 为均匀整数随机变量,对每个比较要重新采样。因为最好的个体被置为最大适应值分数,从而最好的个体能够保证生存下来。

总之,在进化算法的一般框架下,进化规划可以描述为

$t = 0$

初始化 $P(0) = \{a_1(0), \dots, a_\mu(0)\} \in I^\mu$

度量 $P(0); \{\Phi(a_1(0)), \dots, \Phi(a_\mu(0))\}$, 其中 $\Phi(a_K(0)) = \delta(f(x_K(0)), \theta_K)$

while $(\wedge (P(t)) \neq T)$ do

变异: $a'_K(t) = m'(a_K(t)), \forall K \in \{1, 2, \dots, \mu\}$

度量: $P'(t) = \{a'_1(t), \dots, a'_\mu(t)\}; \{\Phi(a'_1(t)), \dots, \Phi(a'_\mu(t))\}$

其中 $\Phi(a'_K(t)) = \delta(f(x'_K(t)), \theta_K)$.

选择: $P(t+1) = S_{(q)}(P(t) \cup P'(t))$

$t = t + 1$

end

6.3.5 进化策略

进化策略的研究始于1964年,当初主要是用于试验处理流体动力学问题,如弯管形态优化。在ES的发展中,Rechenberg和Schwefel做出了重要贡献:Rechenberg对称为 $(1+1)$ -ES的进化策略发展了收敛速度理论, $(1+1)$ -ES是一个简单变异选择机制,它每代通过高斯变异作用在一个个体上来产生一个子代。Rechenberg还首先提出了 $(\mu+1)$ -ES,这个策略是将 $(\mu>1)$ 个个体经过选择形成的一个子代替换掉变异后最差的父代个体。Schwefel在此

基础上提出 $(\mu+\lambda)$ -ES 和 (μ,λ) -ES,前者是针对多父本和子本的,在算法过程中, μ 个父本产生 λ 个子本,这 $\mu+\lambda$ 个候选解同时参加竞争并适者生存(一般生存数目为 μ);后者也是针对多父本和子本的,在算法过程中, μ 个父本产生 λ 个子本,但只有这 λ 个候选解竞争,然后父本被完全取代,也即每个候选解的生命期只限在当前一代。这两种策略(尤其是后者)在进化策略中占据着重要地位。

下面我们将描述它们最一般的形式。

1. 表示法和适应值度量

在进化策略中,搜索点是 n 维向量 $X \in R^n$,个体的适应值等于其目标函数值,即 $\Phi(a) = f(x)$,其中 x 是个体 a 的目标变量部分。此外,每个个体可以包括至多 n 个不同的方差 $C_{ii} = \sigma_i^2 (i \in \{1, \dots, n\})$ 和至多 $n(n-1)/2$ 个协方差 $C_{ij} (i \in \{1, \dots, n-1\}, j \in \{i+1, \dots, n\})$,从而至多 $w = n(n+1)/2$ 个策略参数可以和目标变量组合在一起构成一个个体 $a \in I = R^{n+w}$;不过,一般只考虑方差,从而 $a \in I = R^{2n}$,有时甚至对所有目标变量只用一个共同的方差,这时 $a \in I = R^{n+1}$ 。

2. 变异

在进化策略中,全体 $a = \{x, \sigma\}$ 在变异算子作用下变为 $a' = \{x', \sigma'\}$,其中

$$\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$$

$$x'_i = x_i + N(0, \sigma'_i), \quad i = 1, \dots, n$$

其中 $N(0,1)$ 表示具有期望值为0,标准偏差为1的正态分布随机变量, τ 和 τ' 是算子集参数,分别定义整体和个体步长。

3. 交叉

在进化策略中,交叉算子 $r': I^n \rightarrow I$ 可以按下列方式产生一个个体:

$$X_i = \begin{cases} X_{S,i} & \text{无交叉} \\ X_{S,i} \text{ 或 } X_{T,i} & \text{直接交叉} \\ X_{S,i} + \theta(X_{T,i} - X_{S,i}) & \text{加权平均交叉} \end{cases}$$

下标 S 和 T 指从 $P(t)$ 中随机选取的两个父代个体, $\theta \in [0, 1]$ 为均匀随机变量。

4. 选择

在进化策略中, 选择是按完全确定的方式进行。 (μ, λ) -ES 是从 λ 个子代个体集中选择 μ ($1 \leq \mu \leq \lambda$) 个最好的个体; $(\mu + \lambda)$ -ES 是从父代和子代个体的并集中选择 μ 个最好的个体。虽然 $(\mu + \lambda)$ -ES 保留最优的个体能保证性能单调提高, 但这种策略不能处理变化的环境, 因此, 目前选用最多的还是 (μ, λ) -ES, 研究表明比率 $\mu/\lambda \approx 1/T$ 是最优的。

从上面的讨论, 我们可以得到 $(\mu + \lambda)$ -ES 和 (μ, λ) -ES 的形式描述:

```

 $t = 0$ 
初始化  $P(0) = \{a_1(0), \dots, a_\mu(0)\} \in I^\mu$ 
度量  $P(0); \{\Phi(a_i(0)), \dots, \Phi(a_\mu(0))\}$ , 其中  $\Phi(a_K(0)) = f(x_K(0))$ 
while  $(\wedge (P(t)) \neq T)$  do
    交叉:  $a'_K(t) = r'(P(t)), \quad \forall K \in \{1, 2, \dots, \lambda\}$ 
    变异:  $a''_K(t) = m'(a'_K(t)), \quad \forall K \in \{1, 2, \dots, \lambda\}$ 
    度量:  $P''(t) = \{a''_1(t), \dots, a''_\lambda(t)\}; \{\Phi(a'_1(t)), \dots, \Phi(a''_\lambda(t))\}$ 
        其中  $\Phi(a'_K(t)) = f(x'_K(t))$ 
    选择:  $P(t+1) = if(\mu, \lambda) - ES$ 
        then  $S_{(\mu, \lambda)}(P''(t))$ 
        else  $S_{(\mu + \lambda)}(P(t) \cup P''(t))$ 
     $t = t + 1$ 
end

```

在 ES 中, 候选解的各个组成部分不像在 GA 中被认为是基因信息, 而被看作是染色体行为的特性; 其连接的性态是不确定的, 不管遗传变换发生了什么, 特性的改变(即候选解的各个组成部分的变化)只依赖于高斯变量。

总之,ES 与 EP 相比,有两个主要的区别:

(1) ES 着重于确定性的选择,而 EP 着重于种群中的竞争选择;

(2) ES 将个体类比为编码结构,而 EP 是将种群类比为编码结构;所以,ES 用选择操作产生新的候选,而 EP 则不。

例如,有一实数函数: $F(x):R^n \rightarrow R$,要求求一 n 维向量 X ,使 $F(x)$ 极小。

设 $X = (x_1, x_2, \dots, x_n)$, 以 $n = 3$, $F(x) = \sum_{i=1}^n x_i^2$ 为例,解决的方法是:设种群规模 $N = 30$, 初始随机设定 30 个父本,假设 $x_i \in [-5.12, 5.12]$, 采用 $(1 + 1)$ -ES, 让一个子本只由一个父本产生,产生规则是:

$$x'_i = x_i + N(0, \sigma), \quad \sigma = \alpha e / n^2$$

其中 α 为一比例常数(此例中取 1), e 为父本的误差。

每代中只保留 30 个最好的向量作为下一代的父本。算法可以很快地收敛到唯一的全局极优点。图 6.16 显示出最优向量的优化率。

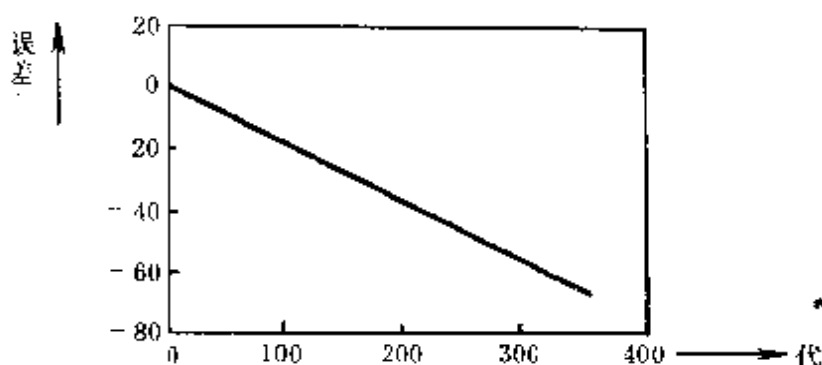


图 6.16 最优向量的优化率

6.3.6 交叉和变异的关系

进化算法研究领域中有争议的问题之一是,遗传算子中是变异还是交叉重要。进化规划和进化策略学派都强调变异算子的重要

要性，并将它作为主要的遗传算子，近来的研究也证实变异算子非常有效，D. Fogel 继续 L. Fogel 的早期的工作，他强烈地声称：在一般意义上，交叉并不优于变异。

在另一方面，遗传算法学派坚信交叉是更有效的算子，在分析交叉和它对性能影响上做了大量的工作，这些研究几乎都认为变异是辅助算子，是次重要的，最近 Schaffer 和 Eshelman 在试验中比较了变异和交叉得出结论：仅有变异并不总是足够的。

遗憾的是，试验上的比较经常是有争议的，并可能误入歧途，人们有必要从理论上彻底解决这一问题。Spears 从定义交叉和变异的两个潜在作用——分裂(disruption)和构造(construction)出发，考虑它们在执行这两个作用上的差异，研究结果表明：对于分裂，变异比交叉有效，虽然它缺少交叉保留个体共同等位基因的能力；然而，对于构造，交叉比变异更有效。

关于变异和交叉相对重要性的问题，可以在更高的层次上来看待，变异用于群体中产生随机多样性，而交叉相当于一个加速器，由部分加速构成整体行为，从而原来的问题就转化为多样性和构造的相对重要性。对于遗传算法，这也与探测(exploration)和开发(exploitation)之间权衡有关。多样性和构造的相对重要性是解答 Holland 体系和 Fogel 体系之间差异的关键。特别地，Fogel 等人怀疑交叉的重要性，他们不相信自然选择会选择个别的特性或特性的组合，交叉被看成是第三位的因素，因为在自然中它似乎不经常出现。

当然，这不一定表明交叉对我们希望求解的问题没用。变异和交叉都不该轻易地提倡和舍弃；每个算子在搜索中起着不同的作用。对于一个问题事先确定哪个算子更重要很难。为得到好的性能在探测和开发之间达到适当的平衡依赖于群体中多样性的数量，应用遗传算法的方式以及所要达到的目标。

总之，标准变异和交叉只是更一般的探测算子的两种方式，现今在交叉和变异之间的区分是否必要尚不清楚。无论如何，设计更

一般的算子将是一条可行之路。

6.3.7 小结

模拟自然进化过程可以产生鲁棒的计算机算法——进化算法。我们在一个统一的框架下对遗传算法、进化规划和进化策略进行了比较，可以发现三种算法既有许多相似处，同时也有很大的不同。进化规划和进化策略都把变异作为主要的搜索算子，而在标准的遗传算法中，变异只处于次要地位；另一方面，交叉在标准遗传算法中起着重要作用，而在进化规划中被完全省去，在进化策略中与自适应结合在一起使用非常重要。另外，标准遗传算法和进化规划都强调随机选择机制的重要性，而从进化策略的角度看，选择是完全确定的，没有合理的根据表明随机选择原则的重要性。进化规划和进化策略确定地把某些个体排除在被选择复制之外，而标准遗传算法一般对每个个体都指定一个非零选择概率。

目前进化算法研究领域中还存在一些有争议的问题，某些非常不同的、有时甚至成对照的设计原则被不同的研究学派所强调。因此，将来研究的一个明确目标将是探明这一事实的原因，从而为设计新的和更好的进化算法找出一般的原则。

参 考 文 献

- [1] 庄镇泉等. 神经网络与神经计算机. 科学出版社, 1992
- [2] 方建安等. 采用神经网络学习的神经网络控制器. 控制与决策, 1993, 3(3)
- [3] 潘卫东. 利用 GA 技术辅助设计 ANN. 模式识别和人工智能, 1994, 3
- [4] Yao X. Evolutionary Artificial Neural Networks, International Journal of Neural Systems, 1993, 4(3), Sep.
- [5] Vittorio Maniezzo. Genetic Evolution of the Topology and Weight Distribution of Neural Networks. IEEE Transactions on Neural Networks, 1994, 5(1), January
- [6] Sietsma J, et al. Creating Artificial Neural Networks that Generalise. Neural Networks, 1994, 4
- [7] Bornholdt S, et al. General Asymmetries Neural Networks and Structure Design by Genetic Algorithms. Neural Networks, 1992, 5
- [8] Peter J Angeline, et al. An Evolutionary Algorithm that Constructs Recurrent Neural Networks. IEEE Trans. on Neural Networks, 1994, 5(1), January
- [9] Herikawa S. A Study on Fuzzy modeling Using Fuzzy Neural Networks. Proc of IEEE, 1991. 562
- [10] Higgms C M. Fuzzy Rule Based Neural Networks for Function Approximation. Proc IEEE IJCNN, 1, 1992
- [11] Karr C L. Design of an Adaptive Fuzzy Logic Controller Using a Genetic Algorithm. Proc of the 4th ICGA, 1991. 450
- [12] Holland J H. Adaptation in Natural and Artificial Systems, Ann Arbor. The University of Michigan Press, 1975

- [13] Fogel L J, et al. Artificial Intelligence through Simulated Evolution. New York: John Wiley, 1966
- [14] Rechenberg I. Cybernetic Solution Path of an Experimental Problem. Roy Aircr, establ, Libr. Trans 1222, Hants, U. K. Farnborough, 1965
- [15] Grefenstette (Ed.) J J. Proceedings of the First International Conference on Genetic Algorithms and Their Applications. Hiusdale, NJ: Lawrence Erlbaum, 1987
- [16] 姚新, 陈国良, 徐惠敏, 刘勇. 进化算法的研究进展. 计算机学报 (待发)
- [17] Grefenstette J J. Optimization of Control Parameters for Genetic Algorithms. IEEE Trans on Syst, Man and Cybern, 1986, 16

第七章 遗传算法与人工生命

人工生命是用人工的方法模拟自然生命的特有行为,而基于遗传算法的进化模型是研究人工生命的主要基础理论之一,因此二者有着密切的关系。本章在讨论了人工生命的研究内容和方法以后,简单介绍了遗传算法与人工生命进化模型共同关心的一些问题。然后着重讨论了L系统的形态生成模型和博弈型人工生态系统,并分别通过对生命行为的模拟和对生物的动作原理的模拟来研究人工生命。最后讨论了人工生命与遗传信息处理,即从信息处理的角度探讨了遗传信息处理模型以及基于这种模型的人工生命合成方法。

7.1 人工生命的研究内容和方法

7.1.1 人工生命及其特征

人工生命是用计算机、精密机械等人工媒体所构造出的能生成自然生物系统特有行为的模拟系统^[1]。这里,“特有行为”主要是指:

(1) 自组织化行为,即不是通过全局的整体控制,而是通过大量的非生命分子(也就是行为的各个构成部分)的相互作用而形成某种有序的行为。

(2) 学习行为,即从生物进化过程的自适应现象中所发现的自学习及其传播行为。

这两个“特有行为”又可以概括为“自律生成行为”。所谓“自律(autonomy)”,含有“自治”、“自我约束”之意,自律系统能够在复杂的外部环境中,自动地调整系统行为,甚至改变系统的结构。在“调整”和“改变”过程中,学习到新的“知识”,使系统本身得到优化。如果

是人工生命系统,则意味着系统得到进化,所以“自律生成行为”以自组织化和学习为基本特征。由于各研究者所处研究领域不同,对人工生命的理解和研究方法也不相同。例如,“强人工生命”和“弱人工生命”就是两个不同的理解层次。“强人工生命”是说,“计算机上的人工生命也能成为生命”,即具有生命行为的计算机可能就是生命体。未来的生物计算机就是“强人工生命”的研究目标。而“弱人工生命”是说,“通过计算机对生命行为的模拟,可以最终形成生命计算理论”,即计算机不会成为生命体,但可以作为研究人工生命的强有力工具,并能表现人工生命的一些特有行为。

从科学的角度和工程的角度来考察人工生命,又可以产生两个不同的研究层次。从科学的角度研究人工生命是“通过对生命现象的基本动力学加以抽象,进行生命科学研究”。而从工程的角度研究人工生命则是“利用生命计算原理研究进化系统和自适应系统的构造方法。”

人工生命的本质就是在人工系统上实现与生物一样的行为^[2]。这里的“系统”,由来自自律的个体集团构成,而个体之间的局部相互作用由简单规则的集合来控制。在这样的系统中,不存在全局范围的集团行为规则。人们观察到的复杂的高维动力学现象及其结构,具有“突生”性质(emergent property),也就是系统不能产生预先设定的性质。因为系统设计者虽然可以设定决定系统中各个个体行为的“局部规则”,但不能预先设定决定个体集团全体行为的“全面行为规则”。这种“突生”性质是由于低维的个体之间局部地相互作用,随着时间的发展而表现出来的。这种性质的产生过程表明,高维结构的“局部层次”,通过要求低维个体的支撑而相互竞争、发展起来。这其中的突生结构,即所谓低维个体行为的组织化完成了极重要的任务,这种任务是通过不断地设定唤起低维个体局部规则而完成的,因此突生结构随时间而进化。

通过上述讨论,我们不难看出人工生命的一些明显特征^[3]:

(1) 人工生命是由单个个体的集团构成,集团中每个个体都只

具有简单过程的行为。

(2) 人工生命系统既不存在全局控制过程,也不存在决定整体行为的规则。

(3) 个体的每个过程都包含与其它个体的交叉,反映了它对局部状态的影响。

(4) 系统能超越各过程范围产生比较高级的行为,并且有“突生”结构与性质。

人工生命的上述特征是所有生物共同具有的特征,这些特征隐含在自然生命现象之中。自然生命当然十分复杂,但概括起来,也可以发现它具有以下一些特征^[3]:

(1) 自增殖是生命产生的最基本特性,也是区别于非生命现象的主要特性之一。

(2) 新陈代谢是生命现象最重要、最基本的活动。它维持生命的存在。

(3) 生命的各部分相互依赖,有机地组成生命整体。

(4) 与环境相互作用,适应、改造环境,使生命得以生存与发展。

(5) 进化是生命存在与发展的具体过程,该过程使生命自身由低级到高级,由简单到复杂,不断演化,不断地完善。

由此看出,生命过程是运动过程。人工生命则是体现生命运动过程的模型化特征,因而它能反映出生命的一些特有行为。

7.1.2 人工生命研究的内容与方法

如前所述,人工生命是用人工媒体产生自然生物的特有行为的系统。由于自然生命现象的多样性和复杂性,使得人工生命研究内容也十分广泛。目前,人工生命还处于模型研究阶段,其主要内容有以下几个方面:

(1) 基于遗传算法的进化模型。

(2) 基于 L 系统的形态生成模型。

(3) 基于博弈理论的人工生态模型。

- (4) 基于神经网络的学习模型。
- (5) 基于自动机理论的自增殖模型。
- (6) 并行分布式处理模型(如邻域模型,孤岛模型等)。
- (7) 免疫系统模型。

上述问题中,遗传算法是进化模型的基本计算结构,是其它问题的研究基础。形态生成、学习、免疫系统等都是进化侧面的基本模型。

由于人工生命与人工智能密切相关,所以有人认为“人工生命形成了人工智能的基本框架结构”^[4];还有人主张把人工生命与人工智能结合起来,通过研究突生(emergence)、进化(evolution)、共生(symbiosis)、多样性(diversity)、情动(motivation)等人工生命的关键课题来研究人工智能^[5]。考虑到人工生命的特征,我们在研究人工生命的上述基本内容时应考虑如下原则:

- (1) 不是自上而下,而是自下而上的研究策略。
- (2) 不是全局控制,而是局部控制。
- (3) 不是复杂行为而是简单行为。
- (4) 不是预先指定行为,而是突生行为。
- (5) 不是个别实体模拟,而是实体集团模拟。

如果能满足上述原则,则可以构划出人工生命研究的大体技术路线,它是一个封闭的环状路线,反映了如图7.1所示的局部与全局的相互依赖关系。该图表明,由局部行为构成全局动力学,再通过全局动力学形成局部环境,这种局部环境对局部行为产生影响,受影响的局部行为再次构成新的全局动力学……。如此反复循环,直至产生人们所期望的“生命特有行为”。

在上述原则基础上,可以从以下两个方面全面展开对人工生命的研究与开发^[6]:

- (1) 基于硬件的人工生命合成。

这里的“硬件”是指以计算机等信息处理

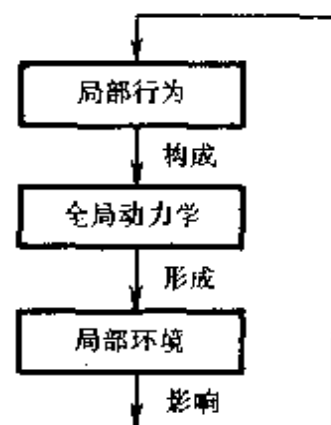


图 7.1 全局与局部的
依赖关系

设备为中心的硬件系统,利用这些系统产生生命行为的方法有两种:一是利用现有的信息处理机器和传动装置来构造能实现人工生命行为的系统(可称之为仿生系统);另一种方法是建立新的生物单元的生命系统,而这种新的生物单元是由实际生物体信息处理机构构成的生物高分子功能体(如免疫抗体、酵素、功能性组织等)。这种基于硬件的生命合成的研究的最终目标是设计出生物计算机,使其表现出人工生命的所有特征和特有行为。

(2) 基于软件的人工生命合成。

这里的“软件”是指用来合成人工生命的计算机软件,它主要包含两个方面:一是基于实际的生物体模型,用计算机软件产生生命行为。这种生物体模型有脑神经系统中的神经网络,有遗传系统中的遗传算法等等。由这些生物体模型抽取数理模型,再在计算机上模拟,以产生模仿生物体功能的人工生命行为。另一个方面是基于动作原理(不存在实际生物体模型)的模拟。生命现象的基本原则是与物理上熵增大原则相反的一种程序化,或者叫自律组织化,产生这种现象的原理就是混沌理论,而基于动作原理的方法就是利用混沌湍流中的分数维、散逸结构等来产生人工生命行为。总之,基于软件的人工生命合成方法很多,以致可能形成如图7.2所示的一种人工生命合成方法论。由图可知,这种人工生命合成方法论的最终目标是产生通用的生命行为理论,使人工生命的研究一般化、理论化。

| | | |
|-------|------------|--|
| 脑神经系统 | 功能结构模型 | 以生物体各功能模型的功能结构为基础建立模型,例如感知机、联想机等 |
| | 功能行为模型 | 对生物体的各功能模型所表现的生命行为建立模型,例如神经网络等 |
| | 行为生成原理模型 | 建立产生生物的生命行为的原理模型,例如自增殖单元自动机、L系统、混沌动力学等 |
| | 组织化实体的行为模型 | 对表现组织化实体的全体行为的各生命行为建立模型,例如博弈理论、遗传算法等 |
| | 通用生命行为理论 | 对生物体所具有的生命行为进行抽象扩展,形成一种通用的理论,这是人工生命研究的最终目标 |

图 7.2 人工生命合成方法论

7.2 遗传算法与人工生命进化模型

遗传算法的起源,与作为神经网络原型的感知机是同一时期。近年来,分子生物学发展迅猛,进展很大,带动了遗传信息科学的发展。在这种背景下,遗传算法得到了更为深入的研究与扩展。目前,对于具有多峰性的复杂系统,利用遗传算法可以给出产生准最优解的实用的最优化方法,它主要应用在工程领域。

遗传算法的基本内容在前面各章已有详细叙述,它采用符号序列来描述信息集合,然后通过一些遗传操作,如交叉(即符号序列的混合)、突然变异(生成符号序列的新的规则)、选择(选取最优符号序列)、淘汰(去除剩余符号序列)等,得到一些优化解。进一步,可以把上述遗传操作反复执行,以得到最优解。人工生命研究的重要内容之一就是进化现象,而遗传算法则是研究进化现象的重要方法之一。若把它与能够分析生命的个体或集团行为的博弈理论结合起来,则可进一步提高人工生命对生态系统的适应性。因此,遗传算法是人工生命研究的重要理论基础之一。

在这一小节中,我们着重讨论人工生命的生成与进化模型和遗传算法关系比较密切的几个问题。为此我们先来看看人工生命的生成结构。

表 7.1 人工生命的生成结构

| 生成结构 | 生成方法 | 例 |
|---------------------|-------|----------------|
| 生物体内部系统 | 建模法 | 神经网络、免疫网络等 |
| | 动作原理法 | 基于混沌、分形的组织化 |
| 生物体外部系统 (实体集团系统) | 建模法 | 遗传算法、博弈理论等 |
| | 动作原理法 | 伴有自组织化的分布式协调原理 |

人工生命的生成结构如表7.1所示。它主要分为两大类:一类是

构成生物体的内部系统,主要包括生物体中的大脑、神经系统、内分泌系统、免疫系统、遗传系统、酵素系统、代谢系统等;另一类是生物实体及其集团所表现的外部系统,主要包含生物实体集团对环境的适应系统和遗传进化系统等。因此,可以从生物内部和外部系统来获取各种各样信息,用这些信息生成人工生命。就其生成方法来说也分为两种:一种是建模法,即先把由内部或外部系统获得的生命行为信息模型化,然后再由这些模型生成生命特有行为;另一种是动作原理法,即基于混沌,分形等原理的生成方法。混沌、分形原理可以用来描述生命行为的原理,因此生命行为是自律分布的非线性行为。

例如直接应用现代计算机技术的人工神经网络系统就是属于生物内部系统范畴的建模系统,而遗传算法则是属于生物外部系统范畴的建模系统。事实上,在神经网络信息处理中,其处理行为就包含着混沌现象。而遗传算法可以被认为是自律分布的并行处理方法。因此人工生命的产生就是从这样一些模型系统所表现出的各种各样生命固有行为出发,把它们的行为原理概括为一些基本算法,用这些算法来生成人工生命。

从以上讨论可以看出,遗传算法可以用来研究生物体外部系统(也就是实体集团系统)中生命行为规律,而这种规律往往表现出自律分布的并行特性。很自然地,人工生命的进化模型与遗传算法有着许多共同研究的问题^[7],这里列举一、二。

1. 个体表现问题

即使是表现相同行为时,某个体如何表现,要决定于该个体所属搜索空间的结构和大小。而这种搜索空间的结构,决定了所谓的“适应度地形”(即淘汰值曲面^[8],该曲面与搜索策略二者决定了进化能力)。一句话,搜索空间和搜索策略决定了人工生命的进化能力。

与个体表现相关的问题还有传感器/效果器的进化能力^[9]。事实上,进化过程中如果没有视觉器官(传感器),也就不会有当今世界各种各样的图像概念;如果没有产生飞行能力的翅膀(效果器),也就不会出现鸟类、有翅昆虫等生物的行为形态。这些例子说明,生物的行

为强烈地依赖于生物所具有的传感器/效果器。因此,传统的方法是保持传感器/效果器系统固定不变,但是传感器/效果器之间的调整器官(对应于大脑的某一部分)可以变化,因而所获得的行为形态受到了很大的限制。在这种情况下,如何表示具有进化能力的传感器/效果器?这是遗传算法和人工生命都面临的一个难题。

2. 搜索策略问题

这个问题也就是在搜索空间内,如何设定搜索点的“转移规划”?在遗传算法中,转移的手段有“交叉”、“突然变异”、“反馈”等遗传操作。这些遗传操作的使用形态(或方式)直接影响搜索能力。在人工生命的情况下,与上述的个体表现问题有关,可以采取这样的方法,对遗传算法中的各种操作,只要适用,就用来表现个体。这样就有可能通过恰当地利用遗传操作,搜索“遗传算法流”。但是在表现个体时如果通常的遗传操作不太适用,则设计者可以凭借经验和直觉“自行搜索”,或者引入新的操作,提高搜索效率。这些操作的能力与淘汰值曲面的形状组合起来,能否使系统离开局部解,决定了它是否具有进化能力。

3. 淘汰与评价问题

通常的遗传算法中的淘汰过程,总是将集团的个体数保持不变。因此,这种过程是全局控制,属于“人为淘汰”,而不是“自然淘汰”,与人工生命中的“局部控制”方针不相符合。因此,人们希望能够通过局部规划来调整个体密度。

更进一步可以考察评价函数问题,最好它也不是由设计者根据外部全面情况给定,而是通过局部相互作用动态地产生,并且事先不明确地予以设定。例如自然界的捕食者与被食者环境中,显然不能说某种动物是绝对的捕食者,另一种动物是绝对的被食者,因为不存在评价“绝对捕食者”的标准。一种动物是捕食者还是被食者,要看其能力是超过还是落后于对手。换句话说,动物能力的评价是一个相对标准,是相对于其它动物而言。人工生命中也是通过在相对标准下个体之间的相互作用来评价个体的。

7.3 L 系统与形态生成模型

L 系统是由美国数学家 A. Lindenmyer 于1968年提出的^[10]。它当时是用来描述红藻(一种植物)生长的一种算法。现在它是用来描述人工生命中生命行为的形态生成原理的算法。L 系统以自动机理论为基础,用符号空间的一个符号序列来表示细胞的状态,把自动机的状态描述为符号序列的状态空间模型。用状态表中的符号序列来表示状态空间中的状态,通过符号序列的变化来描述人工生命的形态生成过程。本节主要讨论植物的型态生成模型。

7.3.1 L 系统与植物形态

我们知道遗传算法是人工生命的基础理论之一,它是模拟生物的优胜劣汰,适者生存的进化过程。通过遗传算法操作,可以进化出一个对给定环境条件适应度最高的个体。把这种思想用在人工智能上,就是找出一定约束条件下问题的最优解的过程。这是一种全局搜索的算法。而人工生命的范畴是很广泛的,同是其基础理论之一的 L 系统则是从另外一个角度着眼,它考虑的是个体的本身。即使是最复杂的生物体,如人,也是由肉眼看不见的受精卵演变而来的,生物体的成长是一个细胞分裂增殖的过程。但是在生物体内总存在一种最基本的信息,这就是基因。它决定了生物体的本质结构及外在形态,同时也存在于生物体的各个组成部分。L 系统正是解析、模拟生物体内这种程序化的自组织,自增殖的行为。不过到目前为止,L 系统主要的、成功的应用是在对低级藻类植物及树木的模拟上,并通过计算机图形的手段得到了大量的栩栩如生的植物形态。L 系统是一种按语法规则来生成图形的数理模型,把它与图形学结合起来则是 A. R. Smith 于1984年及 P. Prusinkiewicz 于1986年的工作。究其本质来说 L 系统是一种形式语言,它最基本的元素是字符与字符串,当然它们可以表示各种各样的物理意义,最基本的操作是循环字符重写,

所以又称字符重写系统。比如令初始字符为 α , 重写的规则为: $\alpha \rightarrow \alpha\beta$, $\beta \rightarrow \alpha$, 则经过数次重写, 字符将变为: $\alpha \rightarrow \alpha\beta \rightarrow \alpha\beta\alpha \rightarrow \alpha\beta\alpha\alpha\beta \rightarrow \alpha\beta\alpha\alpha\beta\alpha\beta \rightarrow \dots$ 。

在下面的讨论中, 我们所说的 L 系统是最简单的一种, 又称为 OL 系统。并有如下定义: 设 V 表示单个字符集, V^* 表示 V 上所有词的集合, 则一个 OL 系统可以表示成 $\{v, w, p\}$, 其中 $w \in V^*$ 为一非空的词, 称为原则, 而 $P \subset C\{v * v^*\}$ 为一有限规则集。若有一对 (c, s) 生成, 则写成, $c \rightarrow s$, 即对每一字母 $c \in V$, 至少有一个词 $s \in V^*$, 使得 $c \rightarrow s$, 当且仅当对每一 $c \in V$ 有唯一的 $s \in V^*$, 使得 $c \rightarrow s$, 则称此时 OL 系统是确定的。

下面就一些植物的计算机模拟来具体介绍一下确定的 OL 系统。

7.3.2 植物的形态生成模型

树有很多种类, 我们一般看看其形态就可以把它们一一区分开来, 我们说有些水墨画唯妙唯肖, 只是它与我们某种内在的感受相吻合, 而决不是指它与哪个具体的树木一模一样。人们的这种内在感受首先来自植物形态的抽象与简化。植物在生长过程中, 为了获得足够的阳光进行光合作用, 必须努力地向上向四周繁殖出更多的树枝、树叶, 使得与阳光的接触表面积最大。如果我们简化这个过程, 则可以像画家作画一样, 采取下列步骤:

- (1) 首先长出一根茎, 茎再向四周长出一些小的树枝。
- (2) 大多数小树枝又长出一些更小的嫩枝, 如此反复。
- (3) 最终每个植物由一大堆枝枝叉叉组成。
- (4) 一棵植物上所有各处都有相似的枝节性质。

在这里, 我们没有考虑花、果实、叶子等等, 也没有引入任何植物学中的名词。它确实很简单, 可能任何一个植物学家都会对这个模型嗤之以鼻。但是这个模型对植物的形态生成, 尤其是它们在视觉上的

表现还是很有意义的。

1. 植物的分枝形式

假如有一个图7.3(a)所示的简单的分枝类型,现在把它数字化成计算机可以理解的形式,如图7.3(b)所示,它由两个最基本元素组成,即单位角度与单位长直线段,为此我们可以用如下的字符串来描述该图。“1[1]1[11]”,我们称“1”为变量,“[”,“]”为标识符。这种ASCⅡ码串很容易被计算机存储及处理,各字符的具体意义如下:



图 7.3 分枝类型及其数字化

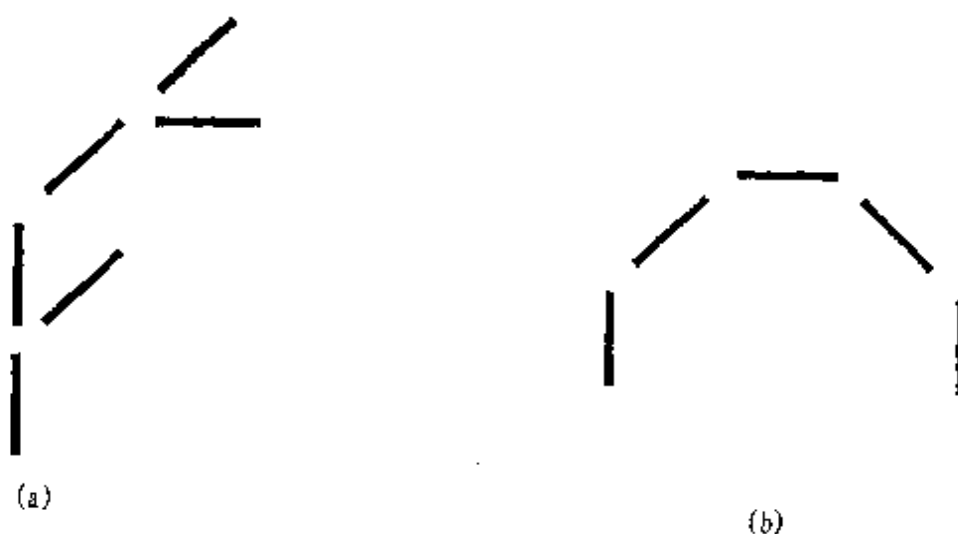


图 7.4 两种分枝类型

“1”表示一个单位的枝长；

“[”表示顺时针转一个角度,此处为 45° ,常表示一个新的分枝的开始；

“]”表示一个分枝的结束。

当然一个分枝上还可以有其它分枝,而某一个分枝还可以连续向一个方向弯曲。例如图7.4(a),(b)的两种分枝类型,可以分别用如下字符串描述“1[1]1[1[1]1]”,“1[1[1[1[1]]]]”。

2. 计算机翻译

所谓计算机翻译,就是根据已知的字符串,在屏幕上画出相应的图形。其基本过程是,首先设定一个起始点 (x, y) 和一个起始方向角度 Angle,然后计算机顺序读取每个字符并翻译、操作。此处各字符表示的操作行为分别如下:

“1”:从当前位置,以 Angle 角方向画出一个单位长的直线段。

“[”:顺时针旋转 45° 角,即将 $\text{Angle} - 45^\circ$ 放回 Angle 中。

“]”:从当前位置回到最邻近的分枝口处,即回到前面又扩大的“[”的地方,同时将 $\text{Angle} + 45^\circ$ 放回 Angle 中。

示意过程如图7.5所示。

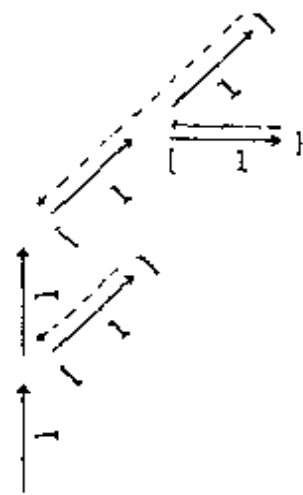


图 7.5 计算机翻译过程

3. 分裂繁殖

上面的例子是很简单的。如果要生成现实中的植物必须包含许许多多的分枝,若仍然是这种从根茎到四周、到顶部的方法——写出符号序列,那将是极端复杂的。实际上每个特定的植物都有一个根本的结构,称之为基本结构,它也可以用字符串描述,植物生长过程可以看作是对基本结构中的每个变量再用相应的分枝类型去替代实现的。在现实生活中,可以看到植物在生长过程中有相似的外在形态,同时,一棵植物的部分与整体之间也有相似的形态。这一点使得我们

的替换过程可以反复迭代进行,也就是第一次迭代将生成一个新的串,第二次迭代是对该串中的所有变量再进行一次分枝类型的替换。这样不仅植物可以生长得高大茂盛,而且在各次迭代阶段也有相同的复杂度和相似度。需要强调的是替换只是针对串中的变量进行的,而对标识符则不作任何替换,直接保留。我们一般把{基本结构,分枝类型,单位长度,单位角度}称作为该植物的基因。它决定了一个特定植物的概念形态,与最终的图形比起来。它的数据量是很小的。图7.6(a),(b)分别为由基因{“1”,11[1[1[1]]],2, $\pi/16$ }分裂繁殖4代、6代的结果。改变基因中的任何一项或多项都可以产生截然不同的形态。表7.2为几种草的基因。

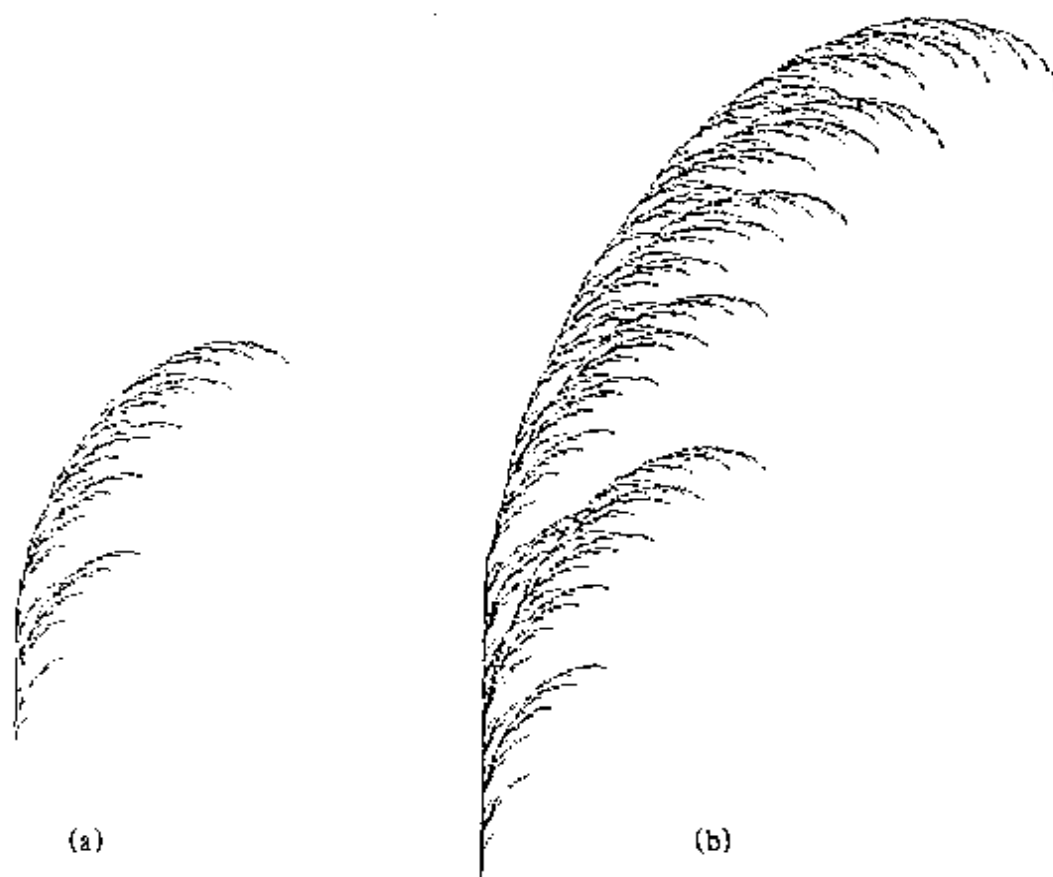


图 7.6 分别迭代4次、6次的形态

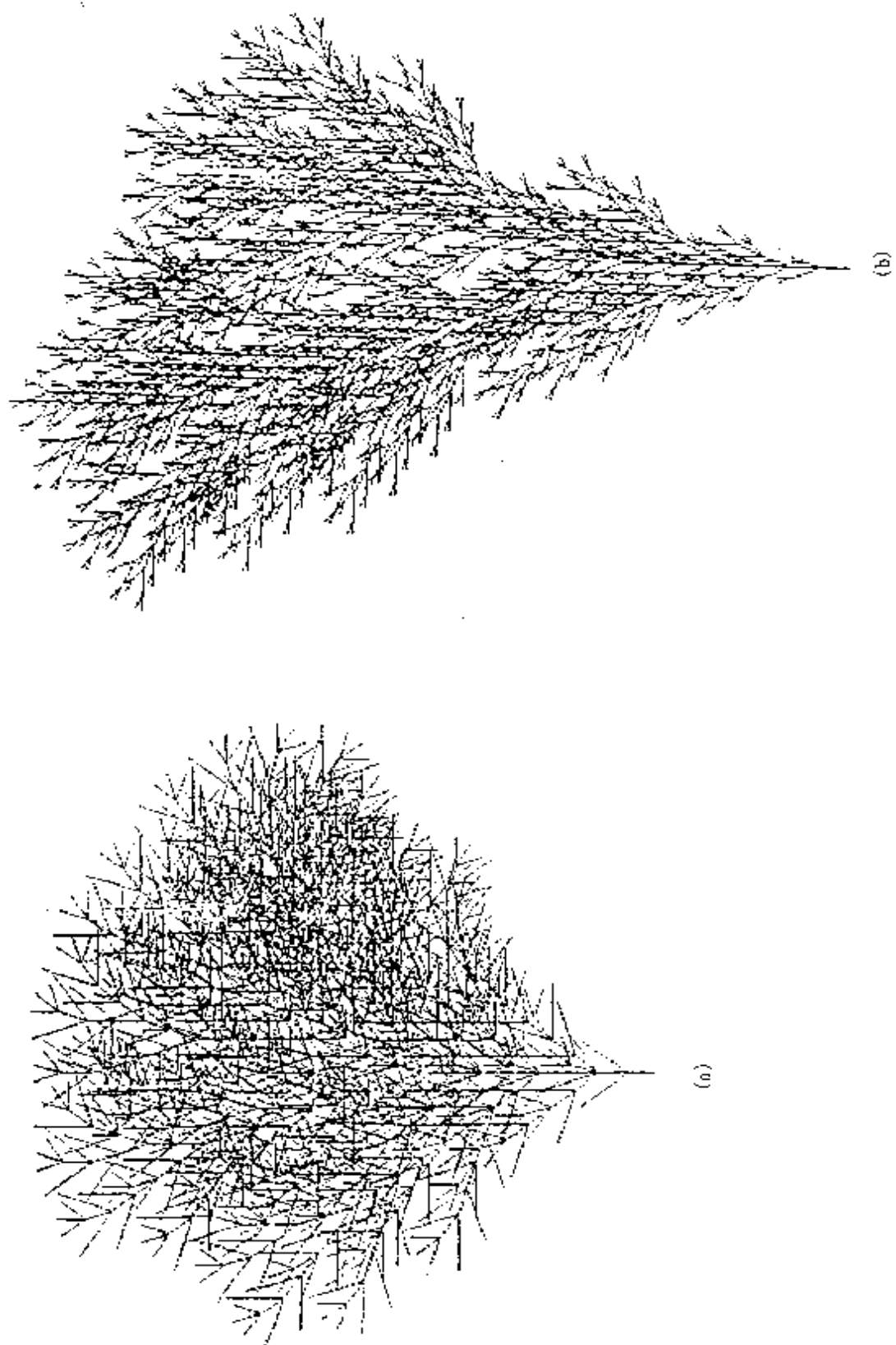


图 7.7 两种基冈生成的形态

表 7.2

几种草类的基因

| 基本结构 | 分枝类型 | 单位长度 | 单位角度 |
|------|-----------------------|------|----------|
| 1 | 11[11[[1]]] | 3 | $\pi/12$ |
| 1 | 1[1]1[[1[[1]]]1[[1]]] | 4 | $\pi/16$ |
| 1 | 11[1[[1]1[[1]]]]] | 4 | $\pi/16$ |

4. 树木

草在风的吹动下,分枝摆向一个方向。而对大树、灌木等,它们的枝枝叉叉可以伸向各个方向,以两个方向为例,我们要在分枝类型中再加两个新的标识符“{”,“}”其中“{”表示逆时针方向上转一个角度,“}”表示回到前一个分枝点处。同时,我们也要增加一些计算机的操作行为:

“{”:角度加一个单位角,例如 45° ,则 $\text{Angle} + 45^\circ \rightarrow \text{Angle}$ 。

“}”:当前位置回到前一个分枝处,且 $\text{Angle} - 45^\circ \rightarrow \text{Angle}$ 。

表7.3为几种有意思的灌木的基因。图7.7(a),(b)分别为表中第4、第3个基因分裂繁殖出的形态。

表 7.3

几种灌木的基因结构

| 基本结构 | 分枝类型 | 单位长度 | 单位角度 |
|------|-------------------------------|------|----------|
| 1 | 1{1{1{1}}[1]}[1[1{1}]] | 3 | $\pi/8$ |
| 1 | 1[1[[1[1[1]]]]7]{1{1{1{1}}}}} | 4 | $\pi/16$ |
| 1 | 11[1{1}1]{11[1]1} | 3 | $\pi/8$ |

5. 进一步扩展

现实中的植物还要比前面的模拟复杂得多。我们可以通过增加基因的内涵来实现更高一级的模拟。在前面讨论的基因中,基本结构串中的变量都只有一个,而每次可用来进行替换的分枝类型也只有一种。其实基因中可以引入更多的变量,相应地可以有更多的分枝类

型。例如我们可以令：

基本结构：“SL111”，单位长：1，单位角： $\frac{\pi}{10}$

对“S”有分枝类型：“[[[G]]]({{G}})TS”

对“G”有分枝类型：“[H{G}L”

对“H”有分枝类型：“{G[H]L”

对“T”有分枝类型：“TL”

对“L”有分枝类型：“{111}[111]1”

其中“S”，“G”，“H”，“T”，“L”为变量，只作替换迭代用。要注意的是，此时的字符‘1’不再是变量，而被视作标识符，它不作迭代替换用，只是在计算机屏幕操作时画出1个单位长的线段。用这种方法，我们绘制了一个小灌木，结果如图7.8所示。

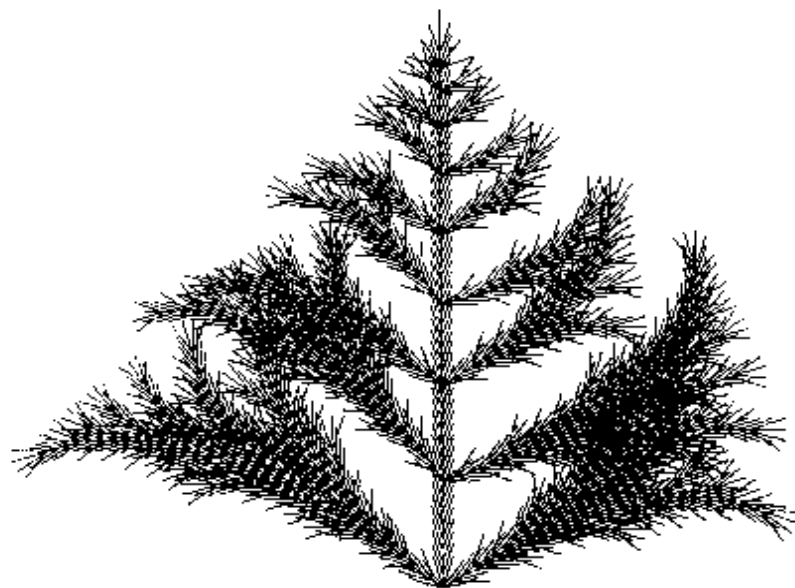


图 7.8 逼真的小灌木

综上所述，我们可以得到一个由 L 系统产生植物与树的流程图，如图7.9所示。

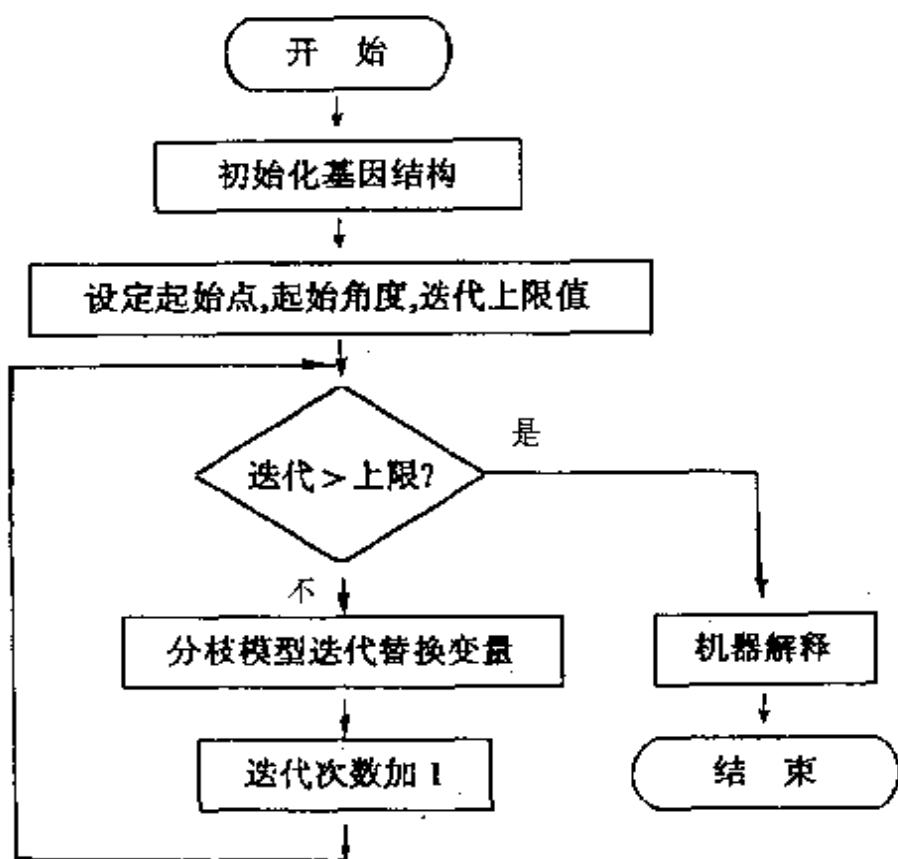


图 7.9 L 系统模拟植物与树的流程图

7.3.3 讨论

(1) 我们前面的模拟没有考虑到树枝和茎的粗细度,这对于蕨类植物的描述似乎还可以,而对树木、灌木等较大形态的植物就不行了,为此可以设想负载越大的枝干,也就是支撑树枝多的枝干应该粗些,这样粗细度就有了一种自然的从高到低递增的感觉。由于现实中的树木其大多数的枝梢都集中在顶部,所以我们可以给单位长度一个缩小的比例来获得进一步真实的模拟。

(2) 为描述的方便。这里的图形都是两维平面上的,但是也可以扩展到三维空间中,此时主要增加的是能反映出空间信息的标识符,同样也可以运用图形学的技术加上合理的色彩、阴影等等。

(3) L 系统一旦给定了,就会得到特定的形体,没有变化。但我们可以非确定的 OL 系统中引入随机性,从而得到若干变形,比

如,对基本结构中的一个变量确定几个相应的分枝模型,利用一定的概率来确定最终将选取哪一个类型进行替换。

(4) 引入自然进化中变异的特征,即在分枝规则中某些字符在替代过程中突然地按照某种策略进行一个小的变化,如变成另外一个字符等等。这种变异对形态的影响以及变异可能的内在机制都是值得深入研究的。

(5) 前面讨论的 L 系统实际上是确定的,上下文无关的,即对每个特定的变量,只用特定的分枝类型串去替换。然而在有些情况下,替代的规则可能要复杂得多。比如根据变量前后字符的意义、关系,采用一定的语法规则进行相关类型的替换。这种系统常常称为 PL 系统(Pseudo L-System)。PL 系统的机制及表现形式是目前研究的一个重点。

(6) L 系统不仅仅可用来模拟植物与树等,它还有广泛的表现范畴。很重要的一个方面是,它可以描述一大类典型的分形集。分形(“fractal”)是美国数学家 B. B. Mandelbrot 于1975年创立的一门几何分支。“fractal”一词来源于拉丁文“fratus”,意为“碎片,破碎”。近数十年来分形的研究得到了极大的关注与发展,它主要用于描述自然界中广泛存在的一大类“奇形怪状”。比如云的轮廓、山的形状、湍流的特性等等,这些形状不再具有欧氏几何可描述的对象的那些基本性质,即平滑规则性(连续可导)。它一般具有比例自相似的结构,表现在两个方面:一是在任何测量尺度下,二是在任何区域局部与总体之间都存在着某种自相似。这种自相似可以是几何上严格的一对一变换,也可以是统计意义上的相似。历史上早已创造出了许多具有严格自相似的分形图形。比如图7.10(a)所示的 Koch 曲线,它的产生过程如图7.10(b)~(e)所示。其方法是把每条线段的中间的1/3部分用同样长度的相交60°角的两条线段替换,并反复操作,最后构成(a)图所示的曲线。可以看到该曲线有一些特别的性质。第一,它有无限的长度,这是因为每迭代一次,总线段的长度增加1/3,同时该曲线与初始直线段又包围一定区域的有限空间。如果初始直线段长度为

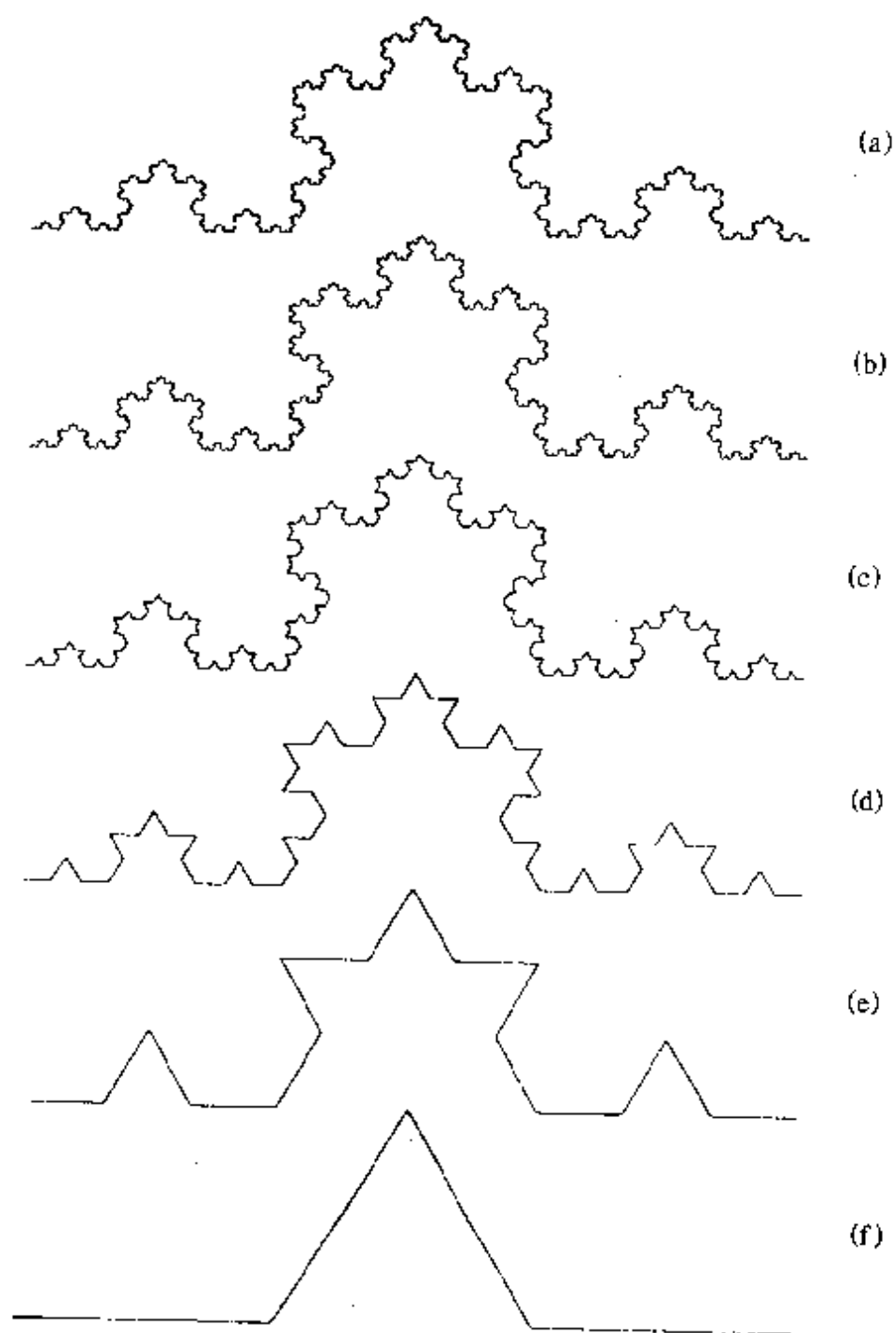


图 7.10 Koch 曲线

1,则可以计算得到所围面积为 $\frac{\sqrt{3}}{20}$ 。第二,曲线上任何两点之间都有无限长的一条 Koch 曲线存在,任何一点处都不可导。用 L 系统能够很容易地产生 Koch 曲线,只要我们令基因为 $\{“1”,\{1}[C1]\{1}”,1,\pi/3\}$ 即可。再如图 7.11 所示的雪花,它的基本结构是由中间一点出发,分出 6 条分枝,两两相隔 60° 角。它由下面字符串描述: $\{(\{11\})\}\{(\{11\})\}\{11\}[11][[11]]11$ 。然后每条分枝再由一个长些的线段加上端点处的两条小分枝所替换,如此反复。雪花的分枝类型为 $“11[1]\{1}”$,单位长度定义为 1,单位角度为 $\pi/3$,它们与基本结构一起,就构成了描述该雪花的基因。

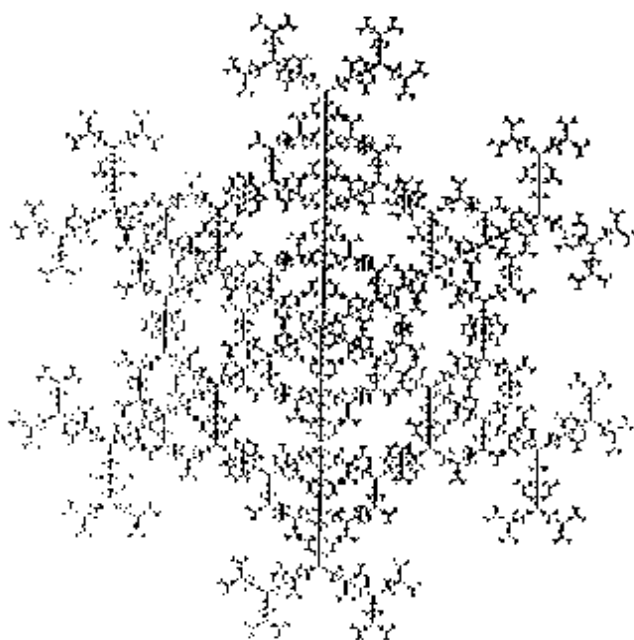


图 7.11 L 系统所描述的雪花

L 系统可以简洁地描述一些严格自相似的分形。而对统计自相似的分形,我们可以通过在 L 系统中引入概率加以描述。我们知道分形同样是人工生命的一个基础理论,它反映了自然界中无限细分、比例相似的哲理,所以又称“大自然的几何学”。生物体中同样到处都有分形的存在。如人的身体表面的穴位与身体的各个器官有着一定的对应关系,只占总体积 5% 的血管可以遍布全身等等。L 系统对分

形结构本身的描述也表明了它对人工生命研究的又一贡献。

对 L 系统的深入研究至少在两个方面是有益的,一是它可以利用计算机等先进的工具生成并能模拟有生命现象的行为;二是有助于从实际的生物体中抽取生命的概念及信息。得出一些有普遍意义的生物信息处理机制,并可为其它的学科领域所利用。

7.4 博弈型人工生态系统

自然界的生态系统以多种自律的个体为构成要素,其中每个个体都属于某个特定的生物种类。每个个体在生存过程中,通过相互竞争而进化。进化过程遵循“适者生存”的普遍规则。

人工生态系统是自然界生态系统的模型化描述。也就是通过建立各种模型,用人工的方法对生态系统进行模拟。在自然生态系统中,每个构成要素都要追求自己的利益,因而在许多场合,当各要素之间发生利害冲突和竞争时,自己的利益与整体的利益往往发生矛盾。因此模拟自然生态系统的一种很自然的方法便是利用博弈理论建立人工生态系统模型,并研究其竞争策略。事实上,在社会生物学,国际关系学以及经济学等许多领域都可以利用博弈理论^[11,12]。但是,到目前为止,大多数研究只考虑了单对博弈的情况,而多对博弈的情况研究得较少。

博弈型生态系统是用博弈理论建立的一种人工生态系统模型^[19],其重要的基础理论当然是博弈理论。在博弈理论中,最有名的是二难推理(dilemma)。它研究了什么是博弈策略最佳解的问题。到目前为止,许多研究者在这个问题上展开了不同层次的研究。有的研究二难推理的最佳策略的搜索^[13];有的研究博弈世界中个体最佳策略与集团最佳策略的差异以及如何克服这种差异^[14];还有的研究“突生”进化的实现机理,探讨策略的进化问题^[15]等等。在本小节中,我们的研究重点是博弈型人工生态系统的自组织化。在简单介绍博

弈理论和博弈策略以后,研究博弈型生态系统原理,然后以二难推理世界为例,讨论生态动力学及其自组织化。

7.4.1 博弈与策略

1. 博弈

博弈世界是博弈型生态系统的模型,其日常行为就是系统各构成要素之间的博弈。最简单的博弈是,针对对手所表示的动作,博弈者采取相应的动作与之相对,并试图获取一定的利益。如果是二人动作的组合,其利益可用利益矩阵表示。博弈的状况包括,博弈者知道多少自己与对手的信息?能否与对手交涉?各种不同场合又是如何考虑?等等。这里,考虑一种非常简单的情况,即对称的 2×2 非合作型博弈。就是一种只有2人参加、2种动作的 2×2 博弈。“对称”是指博弈者与对手获得相同利益的博弈。非合作型博弈是指,博弈者之间所能获得的信息,仅仅是对手所表示的动作,除此之外,没有其它信息。

若两种动作可用 $\{0,1\}$ 表示,则可以有四种博弈状态,即 $(0,0)$, $(0,1)$, $(1,0)$ 和 $(1,1)$ 。令自己的动作为 x ,对手的动作作为 y 时的利益为 a_{xy} ,则从对称性和利益大小的关系,可以有12种不同的情况,其中最著名的博弈就是二难推理,其利益矩阵的元素之间满足下列不等式:

$$a_{01} > a_{11} > a_{00} > a_{10} \quad (7.1)$$

$$2a_{11} > a_{10} + a_{01} \quad (7.2)$$

其它的博弈,如弱虫博弈,其不等式形式与式(7.1)相同,只是把 a_{10} 和 a_{00} 的大小关系翻转一下,即

$$a_{01} > a_{11} > a_{10} > a_{00} \quad (7.3)$$

对于英雄博弈,则有

$$a_{10} > a_{01} > a_{11} > a_{00} \quad (7.4)$$

对于领袖博弈,则有

$$a_{01} > a_{10} > a_{11} > a_{00} \quad (7.5)$$

博弈理论中一般有几种最佳解,如 Nash 解、最大最小解、合计最大

解、ESS(Evolutionally Stable Strategy)等。上述4种博弈的最佳解可大致地用表7.4描述。

表 7.4 代表性的博弈最佳解

| 博弈类型 | Nash 解 | 最大最小解 | 合计最大解 | ESS |
|------|-------------|-------|-------------|-----|
| 二难推理 | (0,0) | (0,0) | (1,1) | 无 |
| 弱虫博弈 | (1,0),(0,1) | (1,1) | (1,1) | 有 |
| 英雄博弈 | (1,0),(0,1) | 无 | (1,0),(0,1) | 有 |
| 领袖博弈 | (1,0),(0,1) | 无 | (1,0),(0,1) | 有 |

对于二难推理型博弈,在相互对抗的情况下,存在 Nash 解和最大最小解,并且由于相互对抗使得双方所得合计利益比较小。在相互合作的情况下,合计最大解与 Nash 解和最大最小解完全不同,产生了进退两难的情况,结果是不存在进化地稳定策略(ESS)。其它种类的博弈,在具有两个不同状态的情况下,都具有 Nash 解。其中弱虫博弈中,最大最小解与 Nash 解不相同,但存在合作的情况,并与合计最大解一致。

以上讨论的是两人博弈的情况。若要同时考虑多个对手的情况

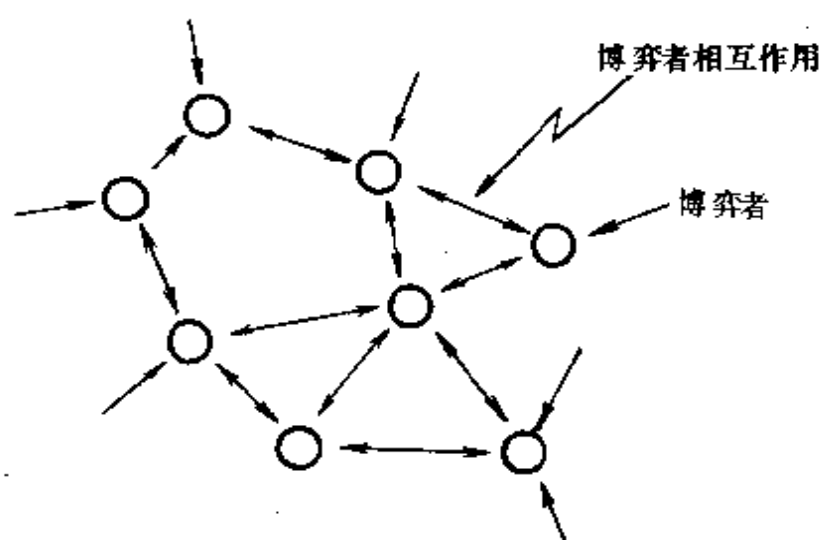


图 7.12 多人博弈模型

就要进行多人博弈。在博弈型生态系统中,当考虑与局部对手的博弈时,往往采用的就是多人博弈模型,如图7.12所示。

2. 博弈策略

博弈策略与博弈方法有关。对于只进行一次博弈与多次反复博弈两种不同的情况,博弈状况会有很大差异。这里介绍的是多次反复博弈并假定博弈过程中能得到的对手信息仅仅是过去所表示的动作。

在这种多次反复博弈方式中,可能的策略种数按某种幂级数快速增长。例如,对于 n 人博弈,当考虑前 k 次自己与对手的动作(向后看 k 步策略)时,策略总数为 $2^{k+2^{kn}}$ 。

(1) 2人博弈($n=2$)的情况。

Axelrod 等人曾于1984年在全世界募集二难推理策略的方案。他们募集到各种各样的策略,并召开了策略竞赛大会^[12]。这些方案按循环赛方式进行优劣竞赛,并按优劣顺序排列,其中比较著名的简单且有力的策略方案有对抗型、合作型、立即还击型等。同时,也有一些由复杂算法组成的策略,但它们的平均成绩并不太好。

(2) n 人博弈的情况($n>2$)。

在多人博弈的情况下,博弈者增加了,博弈的自由度增大了,策略的种数也进一步增加。加上“向后看策略”,就产生了环视周围对手的“环视策略”。环视对手的方法大多数都依赖于博弈对手的空间位置。例如,在二维正方格子空间,最邻近博弈是5人博弈。为简单起见,不考虑对手的位置,只考虑动作总数,则向后看 k 步策略的总数为 $2^{k+2^{kn}}$ 。

为了方便遗传子表示方法的讨论,有必要对策略进行统一描述。我们采用有限状态自动机来表示策略。

3. 策略的自动机表示

描述策略的有限状态自动机 M ,可用一个5元组来表示,即

$$M = (\Sigma, Q, \delta, \sigma_i, q_i) \quad (7.6)$$

其中 Σ 是输入输出动作的有限集合, Q 是自动机状态的有限集合, δ

为转移函数, σ_i 为初始动作, q_i 为自动机的初始状态。转移函数 δ 是根据博弈对手的输入动作 $\sigma_j^k (k=1, 2, \dots, n-1)$ 和自动机的当前状态 q_i 来确定输出动作和自动机的下一个状态, 从而决定了博弈策略。

根据这种向后看1步的博弈动作和博弈者的当前状态, 可以表示所有决定性的策略, 复杂策略的状态数用多个状态机表示。在实际模拟过程中采用只依赖于对手动作总数的策略, 考虑向后看1步的策略。在这种情况下, 策略种类急剧减少为 2^{n+1} 。这样就可以用1个状态自动机来描述。上面的自动机 M 就简化为3元组 M' :

$$M' = (\Sigma, \delta', \sigma_i) \quad (7.7)$$

其中转移函数 δ' 仍然不依赖于自动机状态 q_i , 而只与输入动作总数 $\sigma_j^k (k=1, 2, \dots, n+1)$ 有关, 即

$$\delta' \sim \|(\sigma_j^1, \sigma_j^2, \dots, \sigma_j^{n-1})\| \quad (7.8)$$

这时我们把动作总数定义为 k , 即

$$k = \|(\sigma_j^1, \sigma_j^2, \dots, \sigma_j^{n-1})\| \quad (0 \leq k \leq n-1) \quad (7.9)$$

这样, 这种策略的输出动作 σ_k 与初始动作 σ_i 一共要用 $n+1$ 位动作序列 $(\sigma_i, \sigma_0, \sigma_1, \sigma_2, \dots, \sigma_{n-1})$ 来表示。例如上面的二维正方格空间里 $n=5$, 总共可表示64种策略。

7.4.2 博弈型生态系统

有了上述准备工作, 我们就可以介绍博弈型生态系统的原理了。在博弈型生态系统中, 主要考虑拥有与博弈者集团有关的各种各样策略的博弈者的淘汰与进化。其中遗传算法起着非常重要的作用。这里的遗传算法是广义的, 它考虑了生物界的遗传学和生态学诸概念。

1. 策略种类与遗传子

对于每个博弈者, 首先考虑它所属的种类和它所拥有的遗传子。这里所谓“种类”是指具有同一种策略的博弈者的集合, 所以也可称之为“策略种类”。所谓“遗传子”是指描述策略的一种表示方法。遗传子有多种表现形式, 如文字选择, 有无冗长性等等。为了简单起见, 我们采用有限状态机的位序列来表示遗传子。因此, 不存在冗长性问

题。具有同一遗传子的博弈者属于同一种类。可以用这种遗传子和策略种类表示法来描述生态系统的淘汰和进化的机理。在前面考虑的二维正方格子空间中的5人博弈,在向后看1步的条件下,可具有64种策略。用6位序列即可表示。这种6位序列就是遗传子的表示形式。代表性的策略种类及其遗传子表示形式如表7.5所示。该表是在二难推理的场合,各策略种类所表现的特征。

2. 淘汰规则与遗传子继承

在博弈型生态系统中,博弈者的适应度就是博弈的利益。系统根据博弈者的适应度大小,决定是否淘汰之。所以,这里的遗传算法主要包括淘汰规则和遗传子的继承方法两个方面。

(1) 淘汰规则。

根据适应度所决定的淘汰规则分为两类:

- 1) 根据系统全体所共有的域值进行淘汰;
- 2) 根据局部集团内部的适应度进行淘汰。

第1)类规则相当于认为周围对手对整个环境的影响比较大,第2)类规则相当于根据局部状况决定是否淘汰。有的时候,也可以同时考虑这两种类型的规则。

在第1)类规则中,既有继承域值,也有增殖域值。当适应度比继承域值小时,策略种类因不能继承遗传子而成为空种类。当适应度比增殖域值大时,周围的空种类转换成与自己相同的种类,从而增殖。这是一种严格的淘汰规则。在第2)类规则中可以选择局部集团中具有最大适应度的种类。也可以在局部集团中,根据适应度概率分布来选择适当的种类,此时往往根据适应度比例来确定概率分布。因此,第2)类规则采用的是比较缓和的淘汰规则。

(2) 遗传子的继承。

遗传子的继承也有多种方法。一种方法是把经过淘汰选择出来的遗传子原封不动地继承下来。第二种方法是通过对手与自己的遗传子交叉,继承交叉所得到的遗传子。第三种方法是,利用遗传子变化中的突然变异而使遗传子发生意外变化,从而获得新的遗传子。

在博弈型生态系统中,通过上述广义的遗传算法,淘汰一些策略种类集团,对遗传子实行交叉和突然变异等混合操作,把遗传子继承到下一代去。在下一代,也进行同样的过程。

表 7.5 二难推理情况下的策略种类和遗传子表示

| | | | |
|----|---|-------|--|
| 种类 | 对抗型 | 遗传子表示 | (000000), (1000000) |
| 特征 | 除了初始状态以外,任何情况下都实行对抗策略。通过与对手的对抗,获得较大利益,同一种类的集团相互抗生 | | |
| 种类 | 友好型 | 遗传子表示 | (111111), (011111) |
| 特征 | 与对手经常合作,在各种不同种类混杂的集团中,弱者有可能成为某种程度大小的集团,从而能成为自生集团 | | |
| 种类 | 立即还击型- k 种 ($k=1,2,3,4$) | 遗传子表示 | (110000), (111000) (111100), (111110) |
| 特征 | 发生 k 个以上的对抗时,立即还击, k 的大小程度不受严格的限制。在各种不同种类混杂的集团中,对于周围的复杂变化,容易产生反复对抗。特别当 $k=1$ 时最敏感 | | |
| 种类 | 乖僻型- k 种 ($k=1,2,3,4$) | 遗传子表示 | (000001), (000011) (000111), (001111) |
| 特征 | 发生 k 个以上的对抗时实行合作,除此之外都进行对抗。在同一种类的集团中,给出复杂的输出动作序列 | | |

3. 时间演化结构

在博弈型生态系统中,除了初始一段特别的产生期之外,可以简单地分为两个演化时期,即工作期和淘汰期,如图7.13所示。在工作期,分散开的博弈者通过与近旁对手的博弈,决定各自的适应度后,进入淘汰期。在淘汰期,通过淘汰规则与遗传子的混合作用,继承适应度较高的遗传子,传给下一代,然后再回到工作期,如此反复。在这种博弈型生态系统的演化过程中,一开始必须经过一段设定初始状

态的产生期。

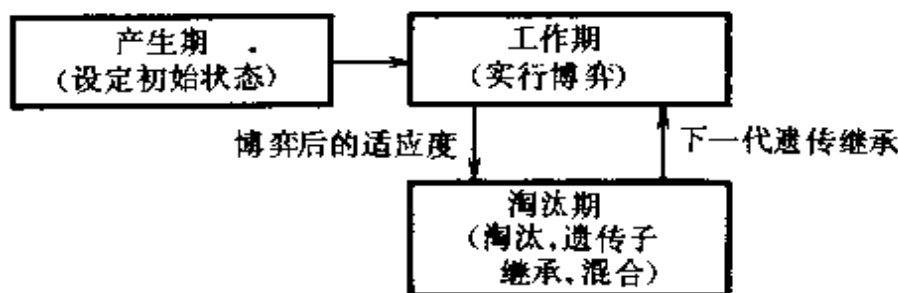


图 7.13 博弈型生态系统的时间演化结构

7.4.3 生态动力学与自组织化

博弈型生态系统经过一段时间的演化,形成了生态动力学,并在这种生态动力学中表现出自组织化。

关于生态系统的自组织化问题,目前还研究得很少。在这里,我们所研究的生态系统自组织化是指,“生态系统中各种策略,根据其适应度,或淘汰,或变异等,缓慢地进化。那些能够适应环境的策略类型或者相互依存的策略类型集团自发地形成具有新的秩序的组织。这一过程就是自组织化过程”。本小节要讨论的是在二难推理情况下的博弈,这种博弈型生态系统也可称之为“二难推理世界”。我们还将用模拟分析的结果来展示其生态动力学的特征。

1. 二难推理世界的生态动力学

在前述的二难推理中,二人博弈的最佳解(Nash 解和最大最小解)所采用的是相互对抗策略。在多次反复博弈的情况下,通过对抗竞赛成绩的比较,说明“立即还击”策略比较强劲^[12]。因此,可以从中探索生态动力学中占优势地位的策略。在博弈者相互之间存在着进退两难利害冲突的二难推理世界中,生态动力学的变化特别丰富。我们来看看它所表现的各种现象。假设在生态动力学模拟过程中,环境参数如下:

- (1) 空间领域:100×100的二维正方格子空间。
- (2) 博弈的反复次数:10次。

(3) 利益: $(a_{11}, a_{01}, a_{10}, a_{00}) = (2, 4, -1, 0)$ 。

(4) 邻近系统: 博弈邻近系统, 淘汰邻近系统。

(5) 淘汰规则: 域值淘汰。继承域值为 θ_1 , 增殖域值为 θ_2 , 且

$$0.1 \leq \theta_1 \leq \theta_2 \leq 5.5$$

(6) 变异: 海明距离为1的遗传子之间的变异

变异率: $0 \sim 0.001$

(7) 遗传子没有交叉。

现在,我们来看看,在上述参数条件下,通过对博弈型生态系统进行基于淘汰域值的模拟,策略集团所表现出的动力学行为特征。

(1) 在特定的淘汰域值(2,4)的周围,由原来的对抗性开始了向合作性的自组织化过程。

(2) 立即对抗-1型($k=1$),产生了免疫抗体,排除了少量抗生种类(弹性的稳定性)。

(3) 在立即对抗-2型($k=2$)占优势的情况下,少量抗生种类原封不动地被接受(塑性的稳定性)。

(4) 乖僻-1, -2, -3型($k=1, 2, 3$),动态地成长,衰退反复进行,并不断延续,在许多情况下都占据优势地位:

1) 即使是少量的,也可以寄生于,或侵略其它类型而成长;

2) 对环境参数的依赖性较小;

3) 通过与其它特定型类进行特别的空间配置,可以展示颇具特点的空间运动。

(5) 从乖僻-3型($k=3$)和对抗型(或者是立即对抗型-1型)的二重层次构造,可以向空的空间演化。

(6) 乖僻-1型或乖僻-3型与对抗型策略具有抗生的寄生关系,所以可以形成整体上占优势的混合集团而表现动力学行为。

从上述结果定性地来看,乖僻型与对抗型的混合策略,在许多场合都处于优势地位,与二人博弈最佳解的结果相符。其中,在特定环境下具有合作性的立即对抗型策略也颇具优势。另一方面,乖僻型策略所得利益波动较大,冒险性较强。

2. 自组织化

在生态动力学中,一般可以把特征的性质定义为“相”。不同的相之间常常相互转换。这些相都具有策略种类的空间分布及其共同的动力学特征。

在二难推理世界中,是通过从对抗性向协调性演化而形成自组织化的。其淘汰域值约在 $(\theta_1, \theta_2) = (2, 4)$ 附近。其策略变化从乖僻种类中以“对抗”优势为中心的特性(“相”)向立即对抗种类中以“协调”优势为中心的特性(“相”)转移,即二难推理世界以表示协调性的总利益为指标,自己组织化为一个有秩序的状态。图7.14是一个具体的生态动力学系统自组织化的过程,它描述了各种族人口的世代变化情况。其中“IS”和“OH”分别表示立即对抗型策略和乖僻型策略的变化轨线。由图看出,乖僻型策略由强渐弱,而立即对抗型策略由弱而强(其淘汰域值条件为) $(\theta_1, \theta_2) = (2, 4)$ 。各相的特征可用世界总人

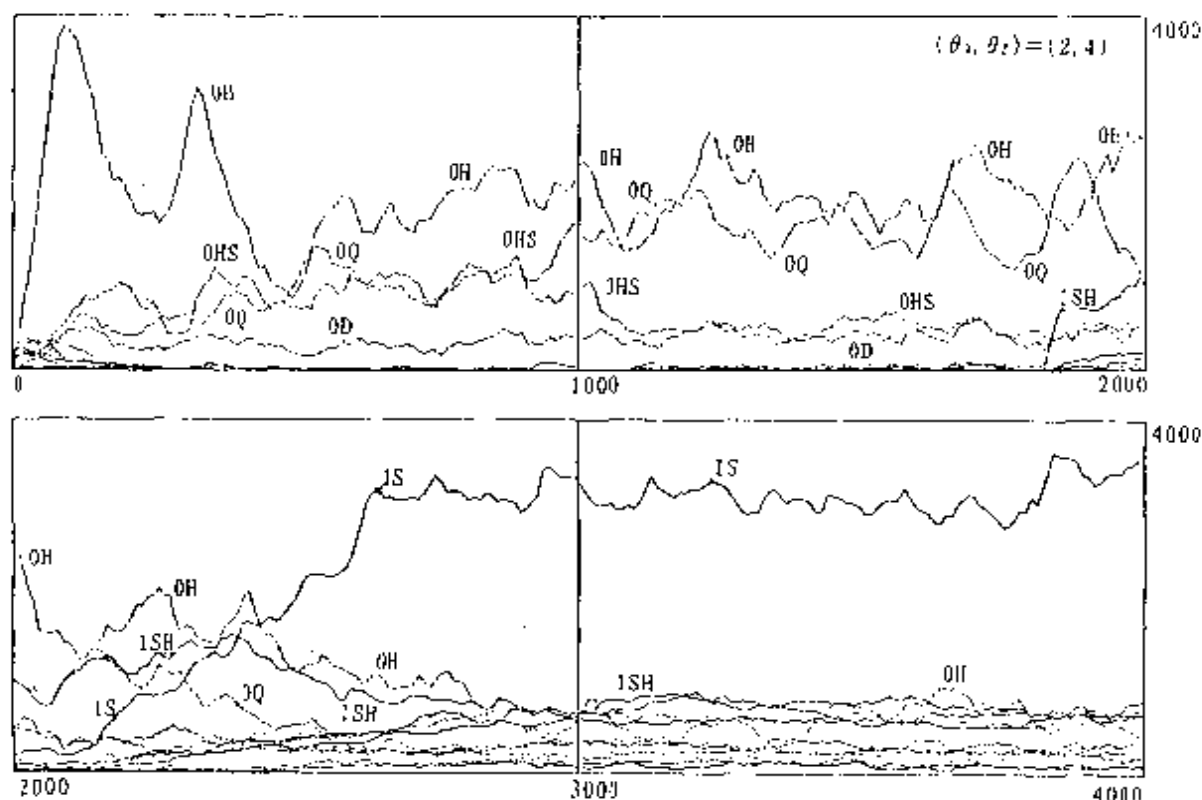


图 7.14 各种族人口的世代变化情况

11和总利益来表示。图7.15表示了世界总人口和总利益的世代变化情况。由图可见,在经过初始的一段随机状态以后,首先到达的相是对抗型种类中具优势的准稳定相(约600~1800代),以后逐渐向立即对抗种类中具优势的稳定相(约2700代以后)转移。两相之间,总人口和总利益明显不同。总利益的变化明显地向协调性方向发展。引起这种相转移的变异,经历了以下一些过程:(000001) (乖僻-1型)→(010001)→(111001)→(111000)(立即对抗-2型)。

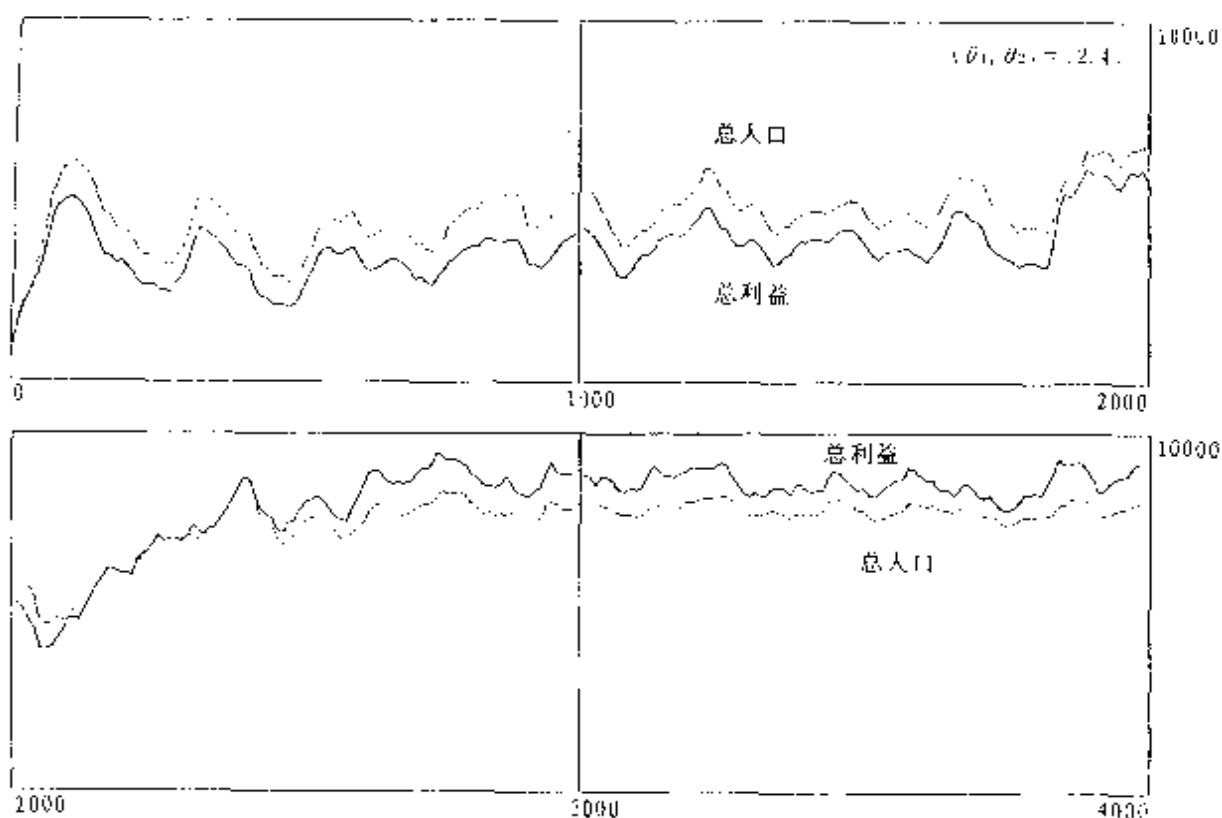


图 7.15 总人口和总利益的世代变化

稳定相与准稳定相的特征的比较示于表7.6,其中种类的熵是用来度量策略种类人口的一种散乱的情形,它的值可以是在某个具有特征的世代(例如,在约600代时,具有准稳定的特征),每相的平均值或标准偏差。

表 7.6 稳定相与准稳定相的比较

| | 对抗相(准稳定) | | 协调相(稳定) | |
|------|----------|-------|---------|-------|
| | 平均值 | 标准偏差 | 平均值 | 标准偏差 |
| 总人口 | 4634 | 462 | 7939 | 189 |
| 平均利益 | 3.25 | 0.096 | 4.38 | 0.069 |
| 种类的熵 | 1.89 | 0.113 | 2.99 | 0.026 |

与对抗相相比,协调相很少在不同世代改变策略种类,其平均利益在增殖域值以上的比率也很大。而对抗相的熵值比协调相小,绝大多数只具有较少的策略种类。另一方面,协调相是多种策略共存。对抗相的各种标准偏差值全部比协调相大,变化也较剧烈。通过以上的一些讨论,我们看到了在淘汰域值的特定范围内如何形成自组织化的。同时需要指出,在这种场合,要使自组织化过程中,立即对抗型策略能成为优势环境,需具备以下一些必要条件:

- (1) 在成长过程中,存在着比某临界范围大的集团。
- (2) 增殖域值不过分地大。
- (3) 变异不过分地多。

但是,如果没有变异,则在有限世代内(例如1600代)系统变成平衡状态,其特征完全由初始分布决定,不会出现自组织化现象。

7.5 人工生命与遗传信息处理

在7.4节讨论的博弈型人工生态系统中,我们指出,通过博弈,可以产生适应环境的新生命。这种新生命是在生态环境中经过“汰劣”以后的“优胜者”。它符合自然界的“适者生存”的普通规律。显然,博弈型人工生态系统是一种生命行为的模型。也就是通过对生命行为的模拟来研究人工生命。而在7.3节讨论的L系统中,我们研究了植物的形态生成方法。事实上是从生物的动作原理的角度研究生命现

象,也就是通过对生物的动作原理的模拟来研究人工生命。在这一节中,我们并不是研究人工生命在信息处理中的各种应用,而是试图从信息论的角度来研究人工生命,探讨生命的信息世界,特别是人类的信息世界。然后讨论遗传监视。在这个基础上再进一步讨论遗传信息处理模型,以及基于这种模型的人工生命合成方法。最后讨论人工生命与人工智能的关系。

7.5.1 人类信息世界

从信息论的角度来看,所谓人工生命就是生命的信息论模型。这个概念意味着,生命的信息结构及其信息处理方法(包括自组织化、进化等)是人工生命研究的核心内容。在这里,我们以世界上最高级的生物——人类为例来探讨生命的信息世界及其与计算机信息结构的关系。

信息的概念最初是用来表现人类的精神活动,由人类创造并体系化而形成的。随着技术的进步,一部分信息开始用机械来处理。人类发明了计算机以后,信息处理的范围大大地扩大了。然而由于人类信息是复杂的、多样的,其表现形式与计算机也有很大差异。因此,用计算机所能表现的人类信息范围仍然受到很大限制。但是,人类总是在不断地探索,试图找出一些新的信息表示方法和处理方法,例如人工智能中的专家系统就是把人类的经验知识用规则的形成表现出来。在人工生命的研究中,人们则是试图用某种方法来描述生命的信息世界。在这里,我们通过人类三种世界的划分及其关系的描述来刻画人类的信息世界。

我们把人类世界分为精神世界、信息世界和物理世界。精神世界是人类的智能活动世界,用来进行问题求解、创造、理解、决策等。信息世界是利用语言、图形等媒体来明确地表现人类所看不见的精神活动的内容,这里的内容是指针对所研究的对象的属性、性质、功能、行为、与其它实体的关系等通过人的感觉器官所感知到的东西,也可以是对它们进一步抽象化所得到的概念及其关系。抽象的概念用来

表现抽象的实体及其复杂功能。而抽象的实体则可以是由已定义的其它具体实体或者抽象实体构成的复合实体。信息世界不仅仅是语言,还包括思维方法、问题求解方法、学习方法等各种智能活动的形式化方法。物理世界是指细胞,DNA 或者微观分子结构等。近年来,人们从神经生理学、脑生理学等领域对生物的物理结构与信息之间的关系进行了许多研究。我们用图7.16(a)来说明这个问题。人类总是不断地进行高度信息处理,而作为物理世界中物质的遗传是人类个体形成的核心。遗传子既是物质,同时也是用来表示遗传信息的语言,因此,也是属于信息世界的一部分。在这种基于信息的物理世界中,通过生理结构的生长而形成个体。正如图中所示的,由物理世界的生体结构、DNA 结构等,通过个体形成过程而产生遗传子,并在信息世界用可能的表示媒体(如语言、图形等)来表示遗传子信息。因此,可以说,信息世界的遗传信息代表了人类生命的信息结构,它在人工生命研究中具有极重要的作用。当然,遗传子信息只是信息世界的一小部分。人类的整个信息世界不仅可以用来表现物理世界的部分内容,也可以用来表现部分精神世界的内容。这样,信息世界作为桥梁把人类的物理结构、行为与人类的精神活动紧密地联系起来了。另一方面,人类的大脑,是由物理世界的大量细胞按一定的结构组成的。它可以实现信息世界的所有功能。但是,到目前为止,人类对其自身的大脑结构并不完全了解,还存在许多未知部分。作为探索这种未知部分的一种方法,人们开始了从结构上研究简单的非生物信息处理功能,如图7.16(b)所示。由图看出,作为非生物系统的物理世界,就是以计算机为核心形成的人工生命系统。在生物系统和非生物系统中,都可以通过简单机理用分层处理的方法实现高度的信息处理功能。但是,所使用的素材不同,其实现方法也有很大差异。在生物系统中,从神经网络内的信号处理开始,如果能通过逐步分层化,使其发挥大脑一样高级的功能,那么不用说信息世界了,就是整个精神世界的实现也将是可能的。

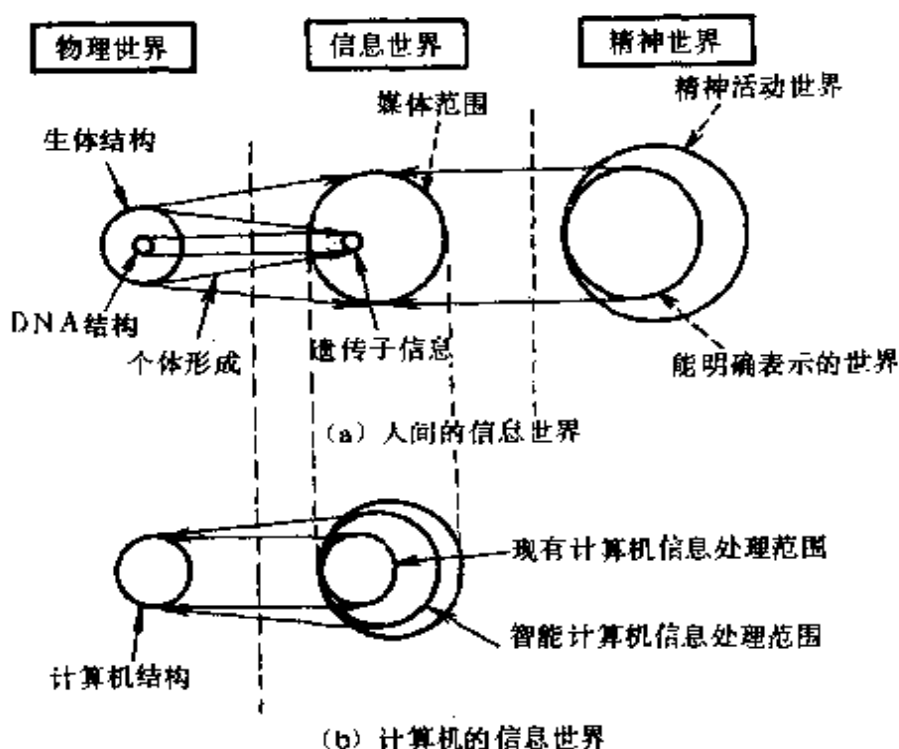


图 7.16 人类信息世界

7.5.2 监视遗传

在构造能适应环境的长期变化而自律行动的系统时,系统的学习目标本身有必要随之变化。系统要学习的目标是十分复杂的环境,这种环境的变化又是由个体集团等的变化等引起的,本身包含着复杂的混沌过程。对于这样的环境,用简单的传统智能系统是难以适应的,因为传统的人工智能系统强烈地依赖先验知识,而在包含混沌现象的复杂环境中,是不存在所谓的“先验知识”的。所以,对于在长期变化的环境下运行的自律系统,需要根据什么基本原理,怎样适应环境等,都是回避不了的问题。在这种情况下,唯一的判断基准就是,系统在这样的环境中能够继续发挥自己的功能,也就是系统具有较好的适应性,使人工生命得以生存。

这种具有适应性的系统的强有力基础之一就是遗传监视理论,如图7.17所示。该理论的主体是自律机构和遗传算法。自律机构主要有两个作用网络,即评价网络和行动网络。评价网络,首先对环境输

入进行评价,产生报酬信号提供给行动网络作为产生行为的评价条件,同时产生一个适应度要求供遗传算法进行遗传操作,以产生个体遗传信息(优良“种子”个体)并传给下一代。行动网络则根据环境的输入、报酬信号和“种子”个体的遗传信息产生特有行为。遗传算法则主要是实行淘汰、增殖、交叉、突然变异等操作,提供优良种子个体以优化个体集团的行为。很多个自律机构在遗传算法的监视(控制)下产生能适应外界环境的整体人工生命行为,也就是个体集团能适应环境变化,继续保持其功能。

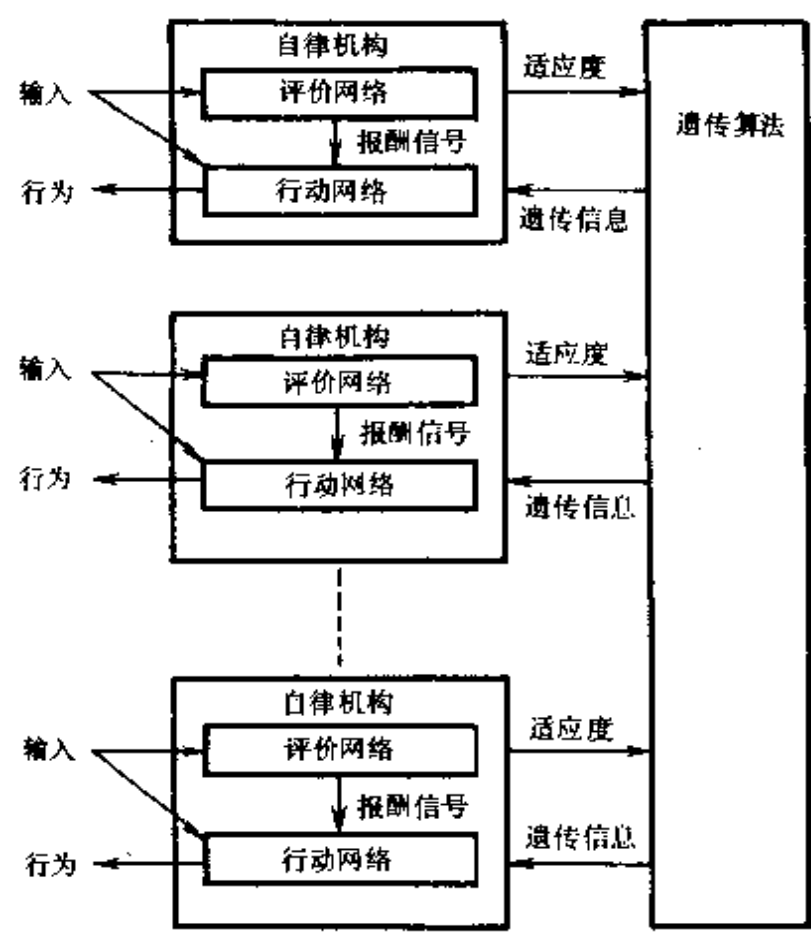


图 7.17 遗传监视

7.5.3 遗传信息处理模型

为了适应环境,生物需要具备一些高级功能。其中由其本身的复杂结构产生的高级功能是生物固有的,也是一般生物系统的特征。生

物系统是一个复杂系统,具有比许多人工系统都优越得多的特性。从根本上说,是生物的生长结构产生了适应于环境的高性能系统。从人工生命研究的角度来说,这一特点是最值得研究,也是最有意义的。

从遗传信息处理的角度来看,生物中的个体形成与生长(或衰退),就是通过遗传子给出生长所需要的基本信息,产生个体的结构;而进化则是为了适应环境的变化,遗传子自身发生变化的过程。两者所需时间也有很大差异。生长时间最短,一般不比个体的寿命长,而进化时间少说也要几代,一般比较长。在计算机内建立的遗传信息处理模型也必须具备生长与进化两个机构。

生物遗传子是用氨基酸的阵列来表现的。这种阵列是一种语言,我们称之为生体语言。该语言的解释是通过化学反应来进行的。遗传、生长、进化等都依赖于遗传子。生体语言的结构十分奇巧,给人工模拟带来了一些约束:硬件语言难以适应这种语言的变化,与人工语言相比,又缺乏表现力。

环境对生物的作用是改变其遗传子。进化,并不是像达尔文学派主张的那样是被动的进化,而是主动的、能动的进化。进化的结构与个体形成的功能密切相关。例如,我们可以假定在每个生命中,每个个体的生长都是非确定的。相同的遗传子有可能生成完全不同的个体。当某个新生个体与环境相适宜时,其性质就可以由后代继承。上述假设的本质就在于,进化并不是总是等待着偶然地产生变异这种完全被动的过程,而是多少有点能动的过程。而且,环境的变化对遗传子的影响也是间接的。

如果某种生物的结构比较简单,则应用上述机理能够产生该生物所有可能的个体结构。如果能够生成2个以上的个体适应于共同的环境,那么,这些表现上不同,但具有相同遗传子的个体,也能通过上述机理产生出来。显然,这种结构只适宜于简单的低级生物,这时遗传信息也比较简单。大体说来,高级生物的结构复杂,其遗传子也复杂,这时,用上述假设的机理就不能产生所有可能的个体,而只能产生其中的一部分。并且,能适应于环境的那一部分个体的概率,也就

是产生变异体的概率,是很小的。我们在建立遗传信息处理模型时,没有必要完全忠实于生物原型。很明显,如果完全忠实于生物原型,仅就其进化过程来说,就得数万年,甚至更长的时间,才能获得希望的进化结果,这是不现实的。

因此,通常是采用以生体语言为代表的人工符号语言。每个个体

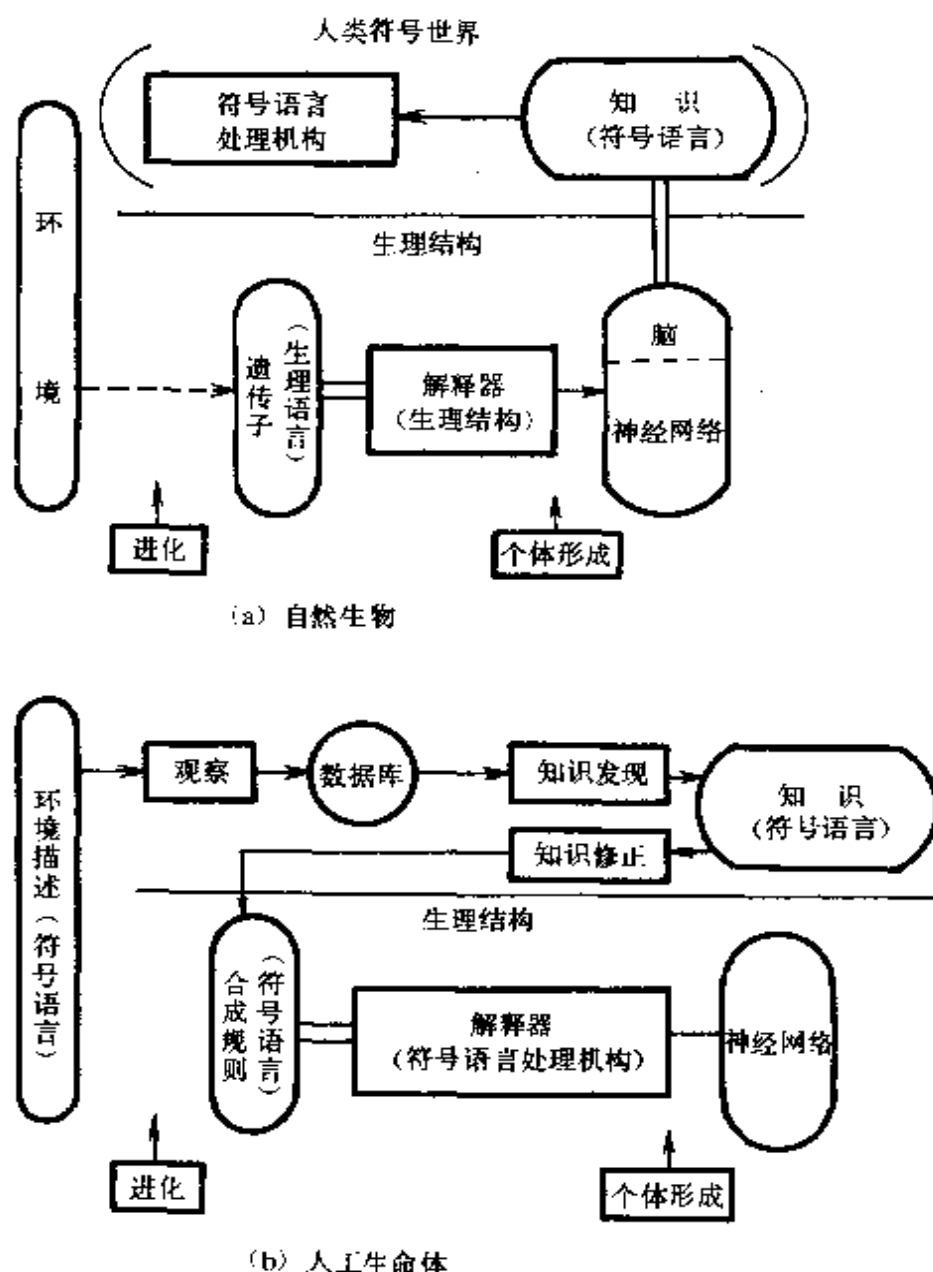


图 7.18 遗传信息处理模型

形成所用的合成规则组,就相当于生物的遗传子,可以用符号表示之。我们知道,在人工智能领域,可以从积累的数据抽取知识。用与此类似的方法,积累所观察的环境数据,通过提炼,抽取规则,更新在处理过程中相当于遗传子的合成规则组。通过不断地实行这一过程,可以完成进化过程。这是一种能较好地适应环境变化的遗传信息处理模型。图7.18所示。该模型围绕着进化和个体形成这两个关键环节,对生物进行模拟。在自然生物系统中,环境对遗传子的作用是直接的。其生命信息的传递与处理是由生物的大脑(由大量的脑细胞构成)和生物本身的生理机构完成的。大脑与人类的信息世界有密切的联系,而人类的信息世界是由符号语言(包括语言、文字、图形等各种信息媒体)来描述处理,并形成知识,输送给大脑的。在人工模型中,生物的遗传子、生理机构,脑神经系统,信息世界等全部模型化。例如用符号合成规则描述遗传子,用人工神经网络模型来描述脑神经系统,外部环境也被符号化。但这里有两个重要特点。一是环境对合成规则(相当于遗传子)的作用是间接的(正如我们前面所假设的),这是对自然生物的一种近似。二是对人类信息世界的描述是用人工智能中基于知识的方法,通过从环境获取信息,加工处理成进化所需要的知识,对遗传子施加作用。由于基于知识的系统具有一定的智能作用和较好的适应性,所以这种人工系统的进化质量较高。

7.5.4 基于遗传信息处理模型的人工生命合成

在人工生命的研究中,除了分析方法以外,还可以采用合成的方法,即合成具有我们所希望的功能、并能适应外部环境的人工生命。在人工生命系统合成的时候,能够直接操作的只有系统的结构。而系统的功能则依赖于其结构。为了寻找适应环境的结构,寻找的方法是很重要的。因此,人工生命的合成问题一般来说是非确定性问题。可以用求解非确定性问题的方法来解决这个问题。通常,人类信息世界中的一般非确定性问题的标准解决方法如图7.19(a)所示。在给定的要求下,通过对初始模型的分析、评价和修正等,实现满足要求的模

型结构,这其中很重要的环节就是反馈,通过反馈来控制模型的结构,所谓控制,也就是对形成个体的合成规则的行为实行制约,这是关于规则的规划,即元规则。而元规则的实现,需要高度复杂的语言。

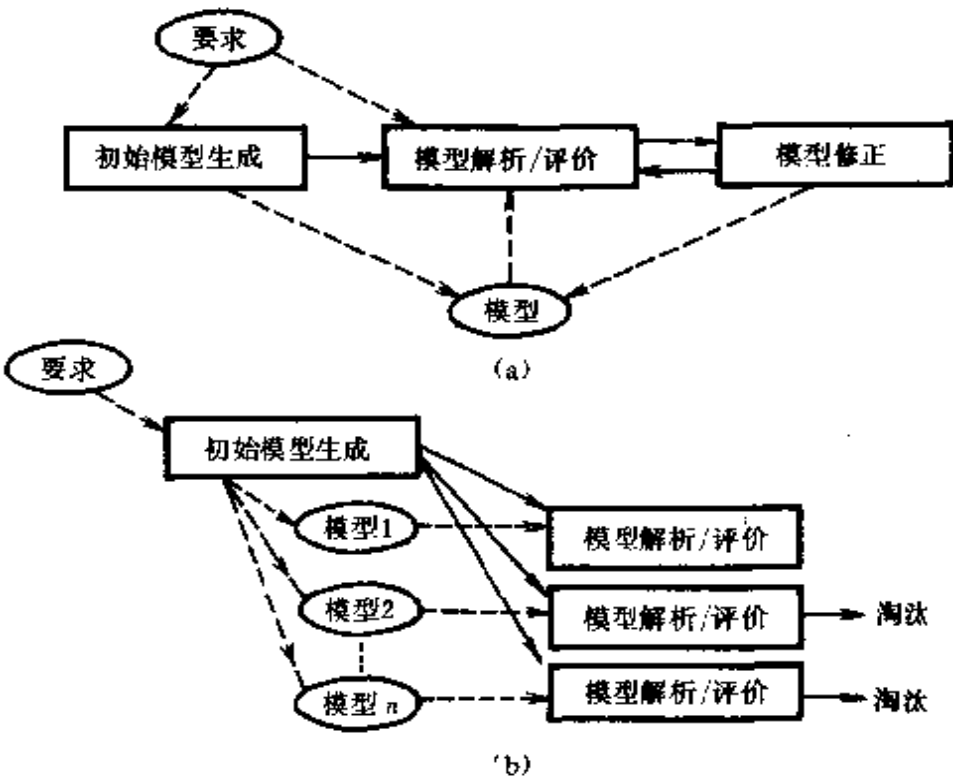


图 7.19 人工生命的合成方法

但是,到目前为止,实际生物采取怎样的方法还不明确,如果不能设计出对生体语言表现力较强的元规则,那么就很难表示含有反馈的过程。因此,在人工生命合成过程中,抓住个体形成与进化这两个关键问题,采用7.19(b)所示的基于遗传信息处理模型的合成方法,它能实现与图7.19(a)相同的功能,但取消了反馈,代之以生物遗传过程中的选择,淘汰方法。即在初始模型的基础上,同时产生多种个体的模型(个体形成),再对每个模型进行分析与评价,选择适应于环境的模型并保存下来,而将其它模型淘汰掉(进化)。这里,对每个模型的评价过程是同时进行的,需要采用并行信息处理技术。

至于合成规则,则是由产生系统结构各个组成要素的指令集组

成。例如“产生细胞”、“产生树状突起”等等。这些指令相互独立,并且通过改变合成规则的产生顺序,可以生成不同的个体。如果随机地选择规划,则通过随机产生的规则的反复作用,也能生成多种个体。当然,这里可以引入一些控制规则,控制规则可以使合成规则的选择稍微合理些。在一些特殊情况下,也可能产生规则选择的一个确定的顺序,此时,由一组合成规则产生的结构是确定性的。这样,根据不同的控制规则的产生方法,结构的产生过程也发生多样的变化。如果控制规则选择适当,结构的产生效率也会比较高。另一方面,在效率高的情况下,系统结构不能适应环境的急剧变化。因此,在急剧变化的环境中,系统结构不能随之进化,从而使之这种生物灭绝。如果控制的元规则能够适应环境的变化,则所产生的个体群也会随之变化,使系统结构对环境变化的柔性增强,这也可视为一种进化。

元规则能否实现,取决于语言。使用人工语言的人工生命是有可能实现元规则的。而生物中的生体语言能否表示元规则,是一个很有意义的问题。与低级生物不同,高级生物很难完全按照图7.19(b)的方式,从信息论的角度考虑个体形成,而是有可能按照介于图7.19(a)和(b)之间的某种形式形成个体。如果是这样,则有必要用生体语言表示元规则。

综上所述,在人工生命系统中,通过简单的合成规则和控制规则的组合,能够产生自律的个体结构。进一步地,如果把这些规则换成一种描述形式,则可以用来模拟进化的结构。通过进化结构的研究,逐步建立更一般的自律模型是我们所期待的。

7.5.5 人工生命与人工智能

在人工智能系统中,一方面要求对各种信息的处理满足足够的柔软性,另一方面又必须用一些固定的简单电路来实现复杂的信息处理功能,并满足各种实际条件的约束。为此,需要把功能分层化,用分层处理的方法降低问题的难度。这种利用简单规则实现复杂功能以及分层处理方法也是研究人工生命中遗传信息处理的一条重要思

路。

在本小节内,我们按照从简单到复杂的顺序,简单地讨论若干个与人工生命和人工智能二者相关的问题,并探讨遗传信息处理的事处智能化方法——智能进化。

(1) 线性可分问题(Linear Decomposition Problem)。这是一个经典的人工智能问题。它所研究的对象可以用其各部分的线性组合来描述。例如,积木问题,可以通过简单化而获得具有完整信息的“积木世界”,它与现实世界的许多问题都具有相同或类似的特性。例如航空公司乘客调度问题。通过把所有乘客、航行路线、所用设备器材等完全符号化,就有可能完整地描述调度规划和约束等问题。在这方面,已经许多成功的自适应专家系统的例子,是经典人工智能的代表。

(2) 线性近似问题(Linear Approximation Problem)。问题本身是非线性的,但在实用中可以在一定的误差允许范围之内。用线性组合的方法近似描述该问题。例如语音处理、自然语言处理、图像处理等属于这类问题。问题是在实用系统中,究竟采用什么方法近似对象的范围,误差范围如何确定等都是非常重要的敏感问题。

(3) 非线性问题(Nonlinear Problem)。从本质上讲,由于问题的非线性特性非常强烈,在实用中不能用线性方法加以近似。是目前人工智能研究的主题,也是人工智能领域非常头痛的问题。问题的非线性如何描述,如何求解等仍是非常棘手的。

(4) 非平衡环境问题(Non-equilibrium Environment Problem)。系统所处的环境非常不稳定,用简单的参数优化方法,不能解决问题,并且即使是具有学习功能的系统也不能解决问题。因为具有学习功能的系统,一般是从环境获得信息,取出需要的部分,根据某种标准进行最优化学习,从而形成“获取信息—优化学习”的组合框架。这是一种动态的学习方法,它运用于环境变化不大的场合。上述“组合框架”相对稳定,能够适应比较小的环境变化。但在许多情况下,环境会发生较大的变化,上述“组合框架”本身也需要变化,就是说,迄今

为止的各种行动模式,都已经不能够适应环境的变化。此时,必须使学习的目的本身也要随着环境变化。获取信息的方法不再是简单的选取某些参数,而是要通过“组合框架(系统)”的自身变化—进化来实现。优化学习的方法也不再仅仅是对参数的优化,而是要通过进化操作(淘汰、增殖、交叉、变异、评价等)产生优化的新信息,这些新信息能比较自然地适应外界环境的变化。例如,某种鸟需要捕捉某条虫子为食物,在这种环境下的最佳行为的学习,就是如何有效地捕捉这条虫。但是,有些时候这条虫也许会产生一些毒素,使环境与以前很不一样,发生了很大的变化。此时的最佳行动战略应该是避开这条虫,去捕捉其它食物。如果是具有学习功能的系统,其学习规则是事先确定的,那么就不能适应上述捕捉虫子的情况。人工生命,特别是进化论的方法,却构成了在这种不稳定环境中构造系统的基础。

上述四个问题中,第1,2两个问题可以用传统人工智能方法解决或部分解决,而第3,4两个问题特别是非平衡环境问题是极为复杂的问题,传统人工智能已无能为力,它的解决需要更高级的智能化方法。根据目前的研究情况来看,构造基于人工生命的自适应智能系统能够改善对复杂的非平衡环境的适应问题。从这个意义上讲,人工生

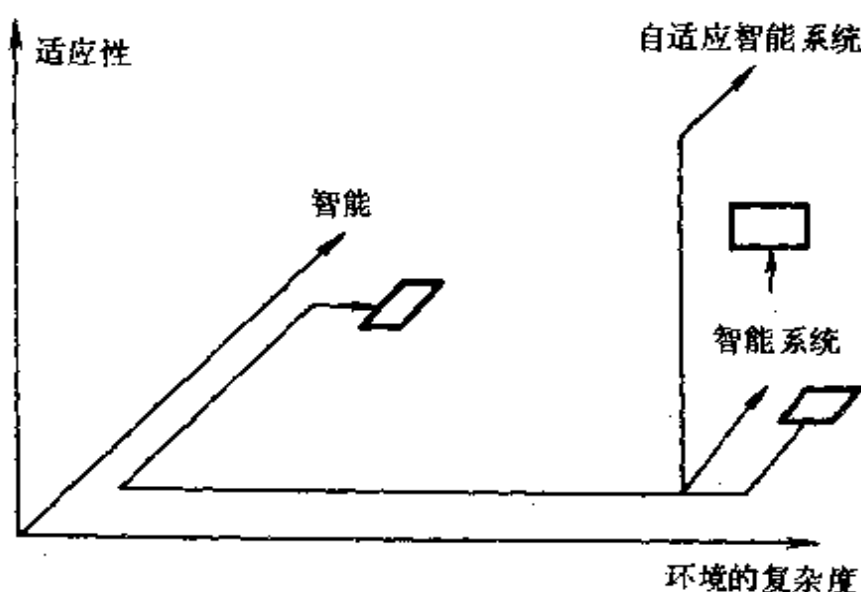


图 7.20 非平衡环境下的自适应智能系统的建立方法

命是一种更高级的人工智能系统,是对人工智能研究的重要发展。当然传统人工智能与人工生命的研究方法是根本不同的,从本章前几节可以看出这一点。

更进一步地,要想真正建立一个能适应复杂非平衡环境变化的智能系统,也并非易事。但建立这种系统的基本思路可用图7.20来表示^[17]。这种思路是以高度适应性为目标,以智能进化等手段,以建立能适应非平衡环境的智能系统。

(5) 智能进化。作为上述第四个问题的解决方法的一个例子,我们来谈一谈智能进化的问题。

在具有智能行为的系统中学习能力的选择是一个很重要的问题。就是说经过选择的学习方法可以被遗传,这对于遗传所产生的后代有重要的影响。但是,如果人工生命的行为能力过分地依赖于遗传子,则当环境发生短期变化时,对系统不利。因此,生命行为有多少份量依赖于遗传,有多少份量来源于对环境的学习,形成了“是遗传还是环境”的问题,也是与神经系统和遗传子的关系有关的问题,在工程上就要涉及到自适应系统的结构问题。我们在本章7.1节就讲过,进化模型是人工生命的最主要研究内容之一。遗传算法是进化模型的基本计算结构,而学习模型等则是进化侧面的基本模型。这就是说高级的进化不仅仅是依赖于遗传,还与学习有很大关系。因此,把遗传算法与学习模型结合起来,研究智能进化是人工生命领域的一个重要课题。

一种简单的智能进化模型就是进化的强化学习(ERL)^[18]。在ERL中,评价网络的结构和结合强度均由遗传决定,而行动网络的结合强度则有可能通过学习决定。而且这种学习信号的源泉,也仅限于评价网络的报酬信号(参见图7.4),这是通过由遗传决定的评价网络的报酬信号对行动网络的结合强度实行最优化的一种强化学习方法。

在遗传监视理论中,最基本的行为、认知等主要由遗传过程决定。而进一步要研究的是由于遗传的影响而获得的行为、认知等。更

进一步则要研究将要获得的那一部分行为、认知等。在这种场合处于非常重要位置的概念的学习则是间接地受遗传监视。比较高级的行为和概念的学习,还要根据已经学习到的知识进行加权,系统的结构成为多层的模型,如图7.21所示。图中强化信号来自于评价网络。

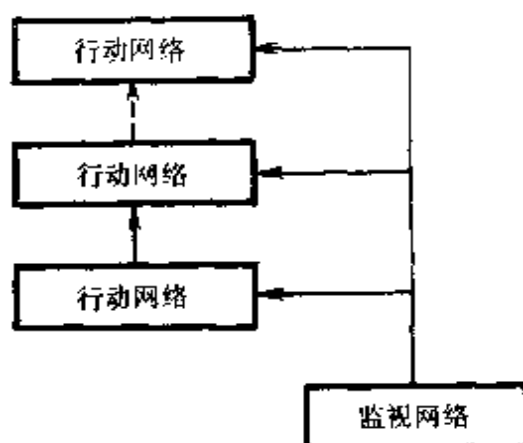


图 7.21 强化学习网络结构

系统中各行动网络的学习按照自下而上的顺序进行,不可能所有行动网络同时进行学习。在实际生物中,已经知道存在着所谓的“时间基因(temporal gene)”,与上述模型是吻合的。当然,这方面的更详细模型还要期待着生物学的研究成果。

显然,强化学习不是智能进化的唯一方法,凡是把遗传算法与学习模型结合起来研究,使人工生物的进化具有智能行为的所有方法都是智能进化研究的内容。

参 考 文 献

- [1] Langton. C G Studying Artificial Life With Cellular Automata. Physica 22D,1986. 120~149
- [2] Langton C G,ed. Artificial Life. SFI Studies in the Sciences of Complexity,1988
- [3] Langton C G. Artificial Life. Addison Wesley,1989
- [4] Belew R. Artificial Life; A Constructive Lower Bound for Artificial Intelligence. IEEE Expert,1991,6(1)
- [5] Kitano H. Challenges of Massive Parallelism. Proc of IJCAI-93, 1993
- [6] 米泽保雄. サイバー・ライフシステム概説 I. Computer today, 1993(55):70~79
- [7] 稻吉宏明. 人工生命への招待. 情報処理,1993,34(7):884~891
- [8] 伏见让. 生物进化を促す2重の散逸構造. 科学朝日,1990-2:18~23
- [9] Cariani P. Some Epistemological Implications of Devices Which Construct Their Own Sensors and Effectors. in ECAL,1991. 484~493
- [10] Lindemayer A, Theor J. Biol.1968,18:280
- [11] Smith M. Evolution and the Theory of Games. Cambridge Univ Press,1982
- [12] Axelord R. The Evolution of Cooperation. BasicBooks, Inc, 1984
- [13] Davis L,ed. Genetic Algorithms and Simulated Annealing. Pitman,1987
- [14] 松尾和洋. シシマ世界における 戦略種の淘汰と発展. 国際情報社会科学研究所研究报告,第16号,1984

- [15] Ikagami T. Ecology of Evolutionary Game Strategy. In: Proc Artificial Life III, 1992
- [16] 松尾和洋他. ゲーム世界における 戦略種の生 ダイナミクス. 国際情報社会科学研究報告, 第26号, 1988
- [17] 北野宏明. 人工生命と人工知能. 日本の科学と技術, 1994, 35 (272): 22~27
- [18] Ackley D, Littman D. Interactions between Learning and Evolution. Artificial Life II, Addison Wesley, 1992
- [19] 北野宏明. 遺伝アルゴリズム. 産業図書株式会社, 1993. 305~322

第八章 遗传算法应用实例

遗传算法的群体搜索策略和不依赖于梯度信息的计算方式,使得它的应用范围甚为广泛。在前面的有关章节中,我们曾详细地介绍了遗传算法在组合优化、机器学习和人工生命等领域中的应用,从而展示出了遗传算法的简单、通用和鲁棒性强的特点。为了进一步扩大遗传算法的应用视野,本章再集中讨论一下遗传算法在图像恢复、图像识别、自适应控制、优化调度和硬件进化等方面的应用实例。

8.1 遗传算法在图像恢复中的应用

8.1.1 引言

恢复退化的图像是数字图像处理的一个重要分支,由于它的理论及实用意义而得到广泛应用。所谓退化图像,是指在图像的产生、传输和记录过程中,由于成像系统的不完善、图像传输介质的影响以及成像系统与被摄景物的相对运动等因素,使得摄取的图像存在有失真及程度不同的变质。经过长期发展,目前已有许多行之有效的图像恢复方法,并形成了相应的理论体系。典型的图像恢复方法有逆滤波法、Wiener 滤波法、奇异值分解伪逆法、Kalman 滤波法、最大熵恢复法和基于 Markov 随机场模型的随机松弛方法^[3]。然而,由于引起图像退化的原因未知或不能用函数表达,使得上述方法要么面临较多的约束问题,要么面临计算求解复杂度的问题。下面我们介绍一种基于遗传算法的图像恢复方法,可以得到较好的恢复结果^[14]。

8.1.2 图像退化模型

图像退化模型可用下式表示^[1]：

$$g(x, y) = \iint h(x, y, x', y') f(x', y') dx' dy' + n(x, y) \quad (8.1)$$

这里 g 为退化图像, h 为退化函数, 也叫点扩展函数 PSF (Point Spread Function), f 为原图像, n 为随机噪声. 当退化函数 h 与具体像素位置无关时, 我们可把式(8.1)改写为卷积形式:

$$g(x, y) = \iint h(x - x', y - y') f(x', y') dx' dy' + n(x, y) \quad (8.2)$$

$$\stackrel{\text{def}}{=} h * f + n \quad (8.3)$$

这里“*”号代表卷积运算。

对式(8.3)采用傅立叶变换:

$$G(u, v) = H(u, v)F(u, v) + N(u, v) \quad (8.4)$$

其中, G, H, F, N 分别表示 g, h, f, n 的傅立叶变换. 利用(8.4)式当 $H \neq 0$ 时可得所求恢复图像为

$$F'(u, v) \stackrel{\text{def}}{=} G(u, v)H(u, v)^{-1} = F(u, v) + N(u, v)H(u, v)^{-1} \quad (8.5)$$

因此无噪声的理想情况下, 有可能完全恢复图像。

8.1.3 遗传算法用于图像恢复

1. 编码

由退化图像本身灰度值直接推测恢复图像, 一个个体就是一幅图像。通常, 基因是一维空间排列, 如用基因来表示二维图像, 它的染色体也是二维的。实验中, 限定图像为二维图像, 二维染色体的位(bit)排列如图8.1。所有值均为1或0。

运用遗传算法, 首先是形成初始集团。我们利用图像恢复的可能

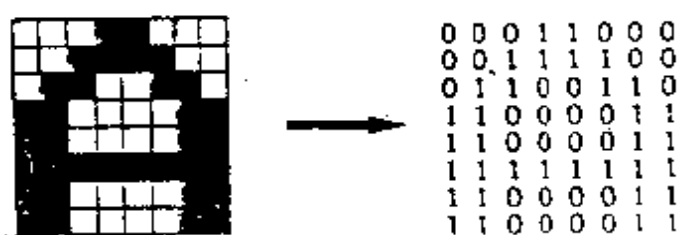


图 8.1 图像基图的编码

解作为初始集团,当然,初始集团的形成可以是随机的。对形成的初始集团,通过选择、交叉和变异等遗传操作使得初始集团不断进化而得到较优解。具体过程及示例见图8.2。

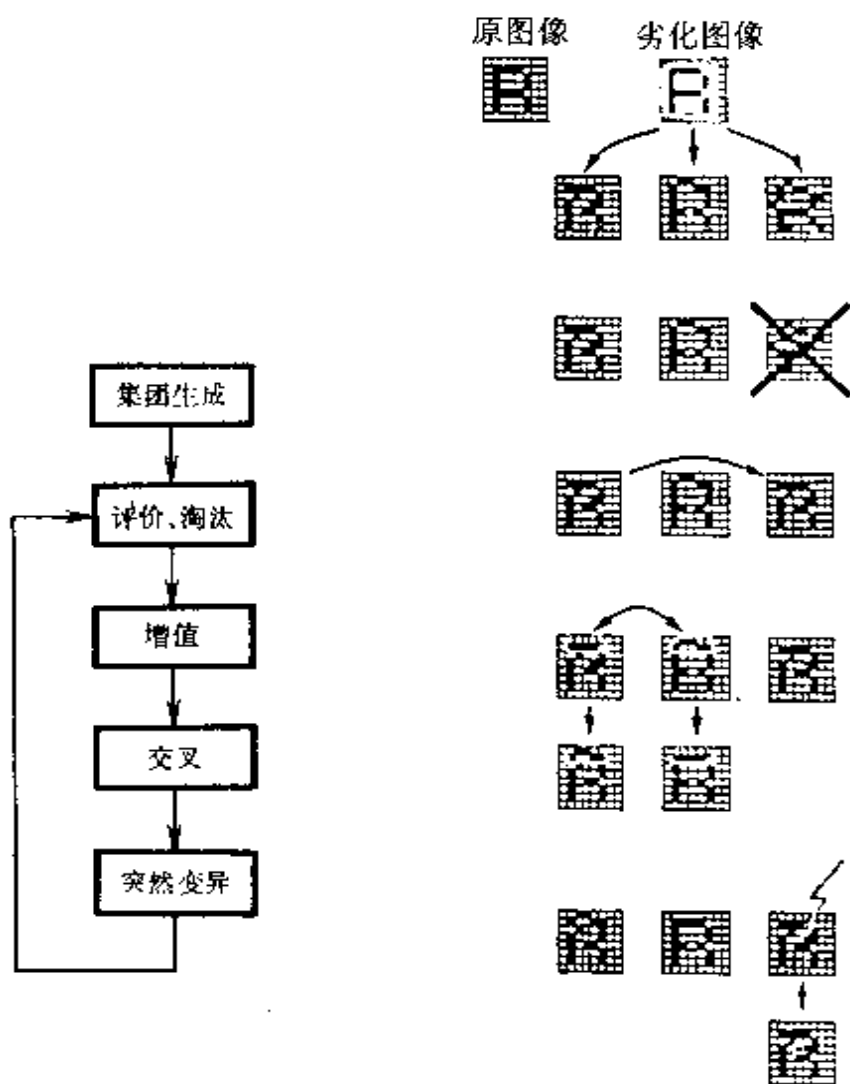


图 8.2 遗传算法应用于图像恢复

需要指出的是,如果退化图像的有关信息能事先观测到并以此为基础来形成初始群体,则可能提高搜索效率。

2. 个体的适应度函数

在遗传操作过程中,要对各个体进行评价,评价函数或适应度函数由下式设定:

$$E(f_i) \stackrel{\text{def}}{=} \|g - h * f_i\|^2 \quad (8.6)$$

这里, f_i 是个体 i 代表的推测恢复图像, g 是观测到的退化图像, h 为退化过程, $*$ 代表卷积。

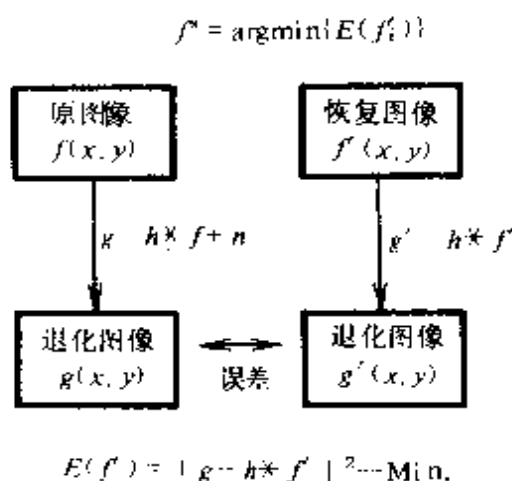


图 8.3 各个体的评价

像,如图8.3。其中 $E(f')$ 的值越小,该个体所代表的图像 $f_i(x, y)$ 的适应度(fitness)越高。最佳图像的恢复过程,也就是最小化 $E(f_i)$ 的过程。 $E(f_i)$ 有时也称为能量函数或目标函数。

3. 遗传操作

对各个体进行评价后,就要对群体进行遗传操作,并按选择、交叉和变异的次序进行。选择的原则是,淘汰适应度较低的个体,保存适应度较高的个体。按照传统的遗传算法,交叉是把两个染色体基因的一段进行交叉,对应于图像,交叉操作如图8.4(a)和图8.4(b)。其中图8.4(a)为按上述原理进行的纵或横交叉而得。但是,考虑到图像处理的特殊性,上述交叉方法的结果不太理想,为此,对交叉操作作一些改进,把两个染色体基因中的多段同时进行交叉,对应于图像的

我们的目的是要求解最佳恢复图像,但最佳恢复图像只能与原图像相比,而这二者均为未知。可是我们知道,退化图像 $g(x, y)$ 是由原图像 $f(x, y)$ 退化而来,退化过程见式(8.3)。因而最佳恢复图像 $f'(x, y)$ 可通过相同的退化过程产生出另一退化图像 $g'(x, y)$, $g(x, y)$ 与 $g'(x, y)$ 可比较,最小化比较结果(即式(8.6))就可得出最佳恢复图

二维染色体,按上述原理进行如图8.4(b)所示的交叉(窗口交叉)。变异操作的实现就是对染色体基因的某一位实施取反操作,在图像处理中,还采用邻近像素平均法控制变异,即在传统的变异操作基础上,再根据该变异位相邻像素的平均值来最后决定该变异位是否改变。按上述原理变异的图像示意图见图8.5(a)和图8.5(b)。

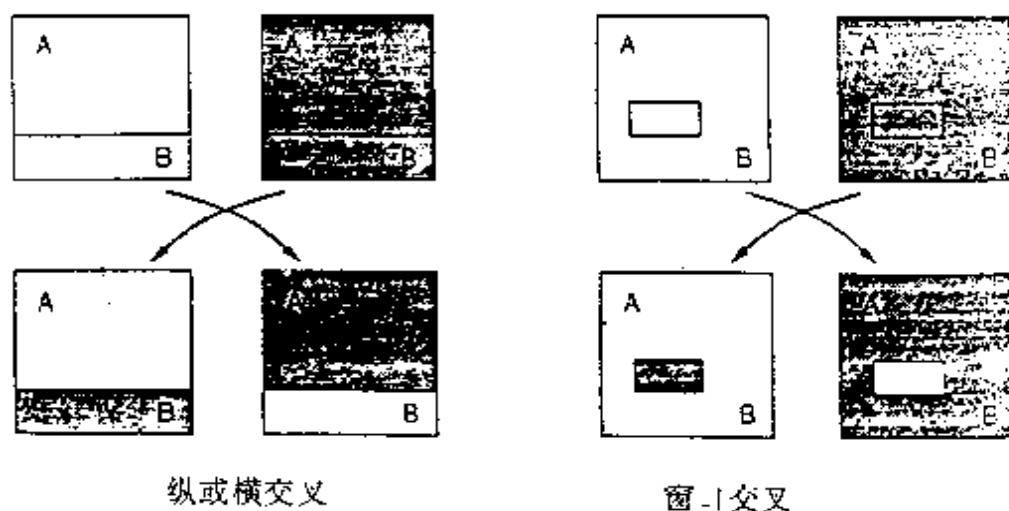


图 8.4 图像恢复的交叉操作

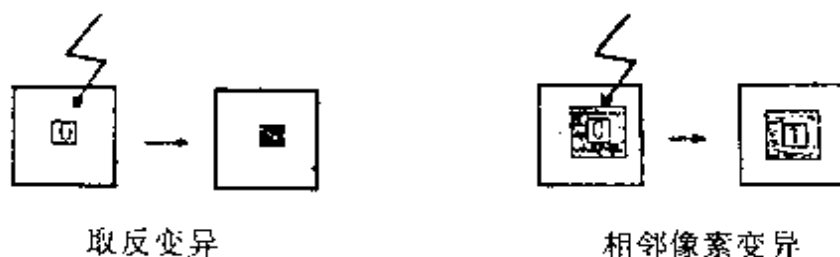


图 8.5 图像恢复的变异操作

4. 实验结果

通过计算机模拟上述图像恢复操作。实验中,选用 32×32 的2值图像,首先经理想低通滤波器进行第一步退化,再加上高斯白噪声进行第二步退化,然后对此退化图像进行恢复。具体的低通滤波器模型为

$$H = \begin{cases} 1, & u^2 + v^2 \leq w_0^2 \\ 0, & u^2 + v^2 > w_0^2 \end{cases} \quad (8.7)$$



原图像



退化图像



恢复图像
(取反变异)



恢复图像
(相邻像素变异)

图 8.6 图像恢复结果(截止频率:4;噪声方差:0.3;个体数:50;世代数:500;图像大小:32×32)

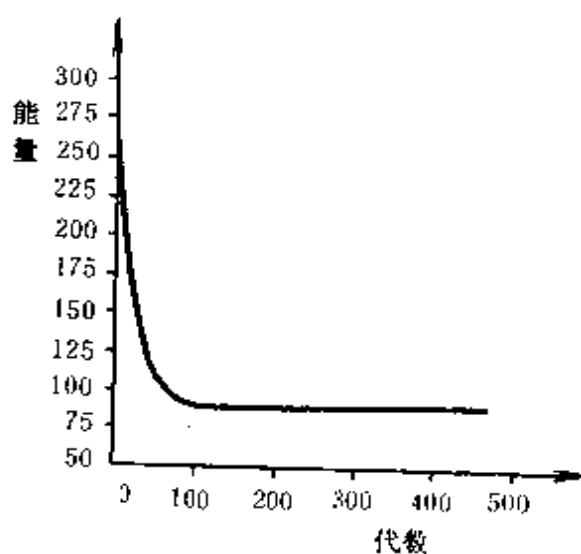


图 8.7 误差结果

这里, $w_0 = 2\pi f_0$, f_0 为截止频率。

图8.6给出了采用上述遗传算法进行图像恢复的结果,并给出了两种变异方法(传统变异和相邻像素平均值变异)的结果比较。我们可以看出,相邻像素平均值变异法结果比传统变异法效果要好。最优个体的评价值与遗传代数的关系见图8.7。交叉概率、变异概率、群体大小等参数的适当选择也将决定遗传进化效率的高低。

8.1.4 遗传算法与贝叶斯方法相结合的图像恢复

前面介绍了单纯采用遗传算法的图像恢复方法,然而,由于式(8.6)作为评价函数没有考虑到图像数据的特殊性,因而其去噪声的效果不佳。我们知道马尔科夫随机场 MRF (Markov Random Field) 能很好地刻画图像数据的空间相关特性,German & German 曾把线过程(line process)引入马尔科夫随机场进行图像恢复,得到较好的去除噪声效果^[3]。

本节介绍一种把遗传算法与马尔科夫随机场模型相结合,并运用贝叶斯方法进行图像恢复的方法^[14]。

1. 马尔可夫随机场模型

由于图像数据的相关性较强,即图像中某一像素的灰度值受其相邻像素灰度值影响较大,而受其较远像素灰度值影响较小,这一特性与马尔可夫随机模型相一致。

German & German 采用了基于马尔可夫随机场模型的贝叶斯方法进行图像恢复,在给定退化图像时,计算原图的最大后验估计(MAP),这个算法是高度并行的,且利用了 Gibbs 分布与马尔科夫随机场(MRF)之间的等价性。该恢复方法的基础是随机松弛算法,它可以产生一系列图像,这些图像在适当的意义下收敛到最大后验估计。由局部的(可能是并行的)像素灰度级的变化和边界元的位置及方向的变化产生这一系列图像。用确定的、逐步改进的方法产生一系列单调增加后验分布的图像。随机松弛同样允许后验分布的随机下降,以防止收敛到局部最大值,该方法实际是一种基于模型的贝叶

斯方法,“模型”由先验分布知识获得。这种模型是分层的,系根据图像的先验知识的类型与程度的不同分别进行处理。假定原始图像用 $X=(F,L)$ 表示, F 是像素亮度矩阵, L 为边缘矩阵。我们把 F 看为亮度过程, L 看为线过程。图像退化模型允许包含噪声、模糊和一些非线性,以表征大多数光化学和光电成像系统。把退化的图像 G 写成形式 $\Phi(H(F))\Theta N$ 这里 H 是模糊矩阵, Φ 是可能的非线性(无记忆)变换, N 是一个独立的噪声场, Θ 表示任何一种适当的可逆运算(如加法和乘法)。为了求解这一问题,我们首先简要地讨论一下亮度过程的马尔可夫性质。类似的讨论也适用于线过程。

用 $Z_m=\{(i,j)\}(1\leq i,j\leq m)$ 表示 $m\times m$ 整数。 $F=\{F_{i,j}\}(i,j\in Z_m)$ 表示原始数字图像的灰度级,小写字母表示变量的取值。例如 $\{F=f\}$ 表示 $\{F_{i,j}=f_{i,j},(i,j)\in Z_m\}$ 。我们把 F 看作随机场的一个样本,通常是各向同性的、均匀的并有较大的相关度。这里我们特别把 F 模型化为 MRF,也就是说 F 的概率服从 Gibbs 分布。给定邻域系统 $F=\{F_{i,j},(i,j)\in Z_m\}$,这里 $F_{i,j}\in Z_m$ 表示 (i,j) 的邻点, (Z_m,F) 上的 MRF 是一个定义在 Z_m 上的随机过程。对任意 (i,j) 和 f 有

$$\begin{aligned} P(F_{i,j}=f_{i,j}|F_{k,l}=f_{k,l},(k,l)\neq(i,j)) \\ = P(F_{i,j}=f_{i,j}|F_{k,l}=f_{k,l},(k,l)\in F_{i,j}) \end{aligned} \quad (8.8)$$

MRF-Gibbs 等价性使得联合概率分布 $P(F=f)$ 可以显式地用能量函数表示。能量函数的选择与 F 十分适合于模型化空间连续性及其它景物特征。

这样,松弛算法可设计为在给定数据 $G=g$ 时,最大化 (F,L) 的条件概率分布,也即找出后验分布 $P(X=x|G=g)$ 的形式。这种贝叶斯估计的形式称为最大后验估计,有时称为惩罚最大似然。

综上所述,图像恢复就是确定使退化图像 g 以原图像 f 为条件(概率)的条件概率 $P(f|g)$ 的最大化,最大后验估计的贝叶斯定理为

$$P(f|g) = \frac{P(g|f)P(f)}{P(g)} \quad (8.9)$$

如果引入图像能量的概念,最大化 $P(f|g)$, 就是使图像的能量最小。对原图像 f 引入马尔可夫随机场模型, 这样, 原图像 f 的概率分布 $P(f)$ 可用原图像的能量 $E(f)$ 来表示:

$$P(f) \propto e^{-\beta E(f)} \quad (8.10)$$

这里 β 为一常数。

另一方面, 退化图像 g 的概率分布与噪声分布有关, 我们把噪声方差 σ^2 引入图像退化模型(见式(8.1)), 得到:

$$P(g|f) \propto e^{-\|g-h*f\|^2/2\sigma^2} \quad (8.11)$$

因而,

$$P(f|g) \propto e^{-(\beta E(f) + \|g-h*f\|^2/2\sigma^2)} \quad (8.12)$$

这样, 我们求解的问题就可归结为最大化式(8.12)。

2. 线过程

前面我们已经说过, 图像可表示为像素和线元素的集合 (F, L) , 这样, 在表示自然图像时既保持了其连续性, 同时, 也保持了其不连续性, 考虑到这部分不连续性, 能量函数 E 可定义为

$$\begin{aligned} E(f', l) \stackrel{\text{def}}{=} & \frac{1}{2\sigma^2} \|g - h * f'\|^2 \\ & + \sum_i \sum_j (f'_i - f'_j)^2 (1 - l_{ij}) + E_{\text{line}}(l) \end{aligned} \quad (8.13)$$

这里, i 为某格点, j 为与 i 相邻的格点, l_{ij} 是随机变量, 格点 i 和 j 之间不连续时, 取值为1, 连续时为0。上式中右边第一项为推测图像 f' 的退化模型图像 $h * f'$ 与退化图像 g 之间的一致性的能量度量; 第二项是推测相邻两点 i, j 之间边界的存在与否, 这是考虑到图像数据的相关性; 第三项是考虑到图像不连续性与连续性之间的相互作用的惩罚项。具体的 E_{line} 可用下式表达:

$$\begin{aligned} E_{\text{line}}(l) = & C_p \sum_i \sum_j h_{ij} h_{i,j+1} + C_c \sum_i \sum_j h_{ij} \\ & + C_L \sum_i \sum_j h_{ij} [(1 - h_{i+1,j} - v_{ij} - v_{i,j+1})^2] \end{aligned}$$

$$+ (1 - h_{i-1,j} - v_{i-1,j} - v_{i-1,j+1})^2] \quad (8.14)$$

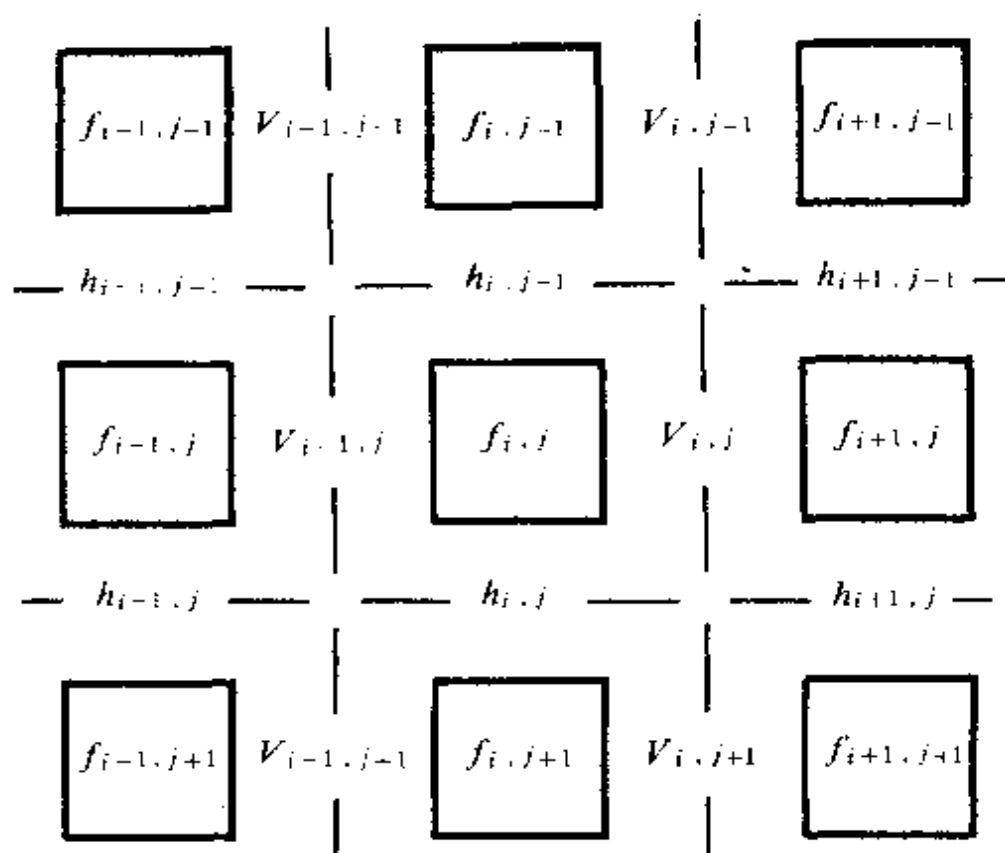


图 8.8 线过程

这里, h_{ij} , v_{ij} 分别代表水平、垂直的线处理。如图 8.8。上式右边第一项为考虑到方向的线处理, 第二项是考虑到本身的线处理的惩罚项, 第三项为考虑到线之间相互作用的能量。各项系数较难确定, 因此, 有必要进行适当的人为能量规定: 如没有线情况时为 0, 结束点为 2.7, 转弯为 1.8, 连续性为 0.9, 分岔为 1.8, 交叉为 2.7 等。如图 8.9。

3. 线过程的引入

用遗传算法进行图像恢复, 引入线过程将会大大改善其恢复结果, 能有效地去除噪声, 这也是因为线过程是图像特征的一个反映。图 8.10 给出了线过程引入前后图像恢复的比较。

4. 空间变化为主因的图像恢复

对于因空间变化而引起的图像退化过程的图像恢复用上述算法

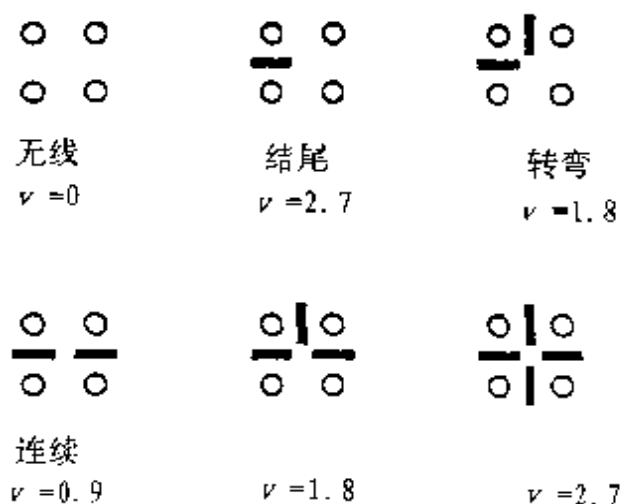


图 8.9 线过程的能量

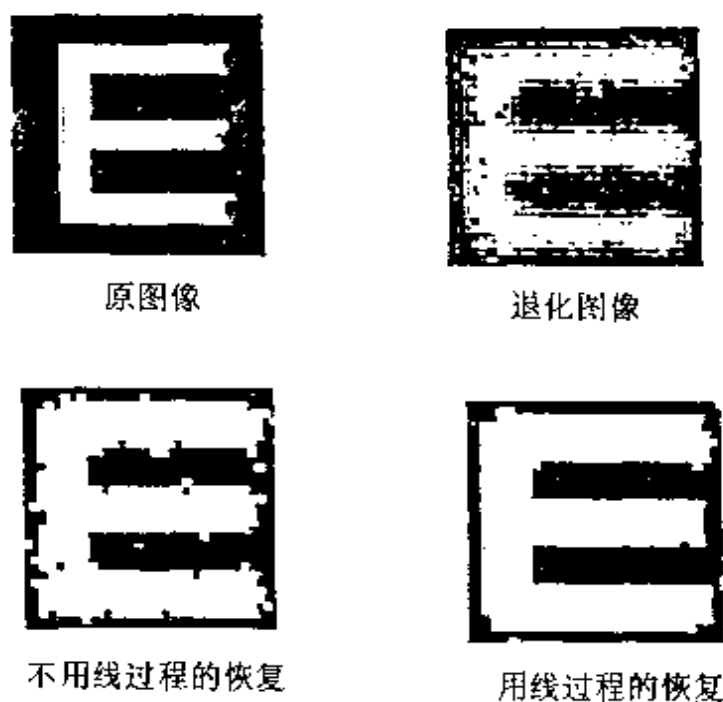


图 8.10 线过程的引入结果

(截止频率:4;噪声方差:0.3;个体数:50;世代数:500;图像大小:32×32)

也可以求解。结果见图8.11。这里,退化函数的形成以图像中心与摄像机焦点的连线为轴按一定的角速度旋转,离图像中心距越远的地方退化的越强,其过程用下式表示:

$$h(x, y, x', y') = \frac{1}{r_0} \delta(r - r_0) \text{rect} \left[\frac{\theta - \theta_0 - \Delta\theta/2}{\Delta\theta} \right] \quad (8.15)$$

这里, $\delta()$ 为 delta 函数, $\text{rect}()$ 为单位直角函数, r, r_0, θ, θ_0 用下面公式定义:

$$r = \sqrt{x^2 + y^2}, r_0 = \sqrt{x'^2 + y'^2},$$

$$\theta = \tan^{-1}(y/x), \theta_0 = \tan^{-1}(y'/x')$$



原图像



退化图像



用线过程的恢复

图 8.11 空间变化为主因的恢复结果
(噪声方差:0.1;个体数:50;世代数:500)

8.2 遗传算法在图像识别中的应用

8.2.1 引言

图像识别是计算机视觉中非常广泛的研究分支,识别中,噪声的影响将直接反映在识别的结果上,所以一般研究图像识别时噪声将是一个不可忽视的因素,这也是计算机视觉走向应用的一个难题,比

如雷达系统在以“偷窃”方式工作所获取的图像、医学成像系统在以最小辐射损害方式所获取的图像,均是低信噪比图像。在传统的“垂直集成”视觉系统中,其处理从原始传感器数据开始,到高层任务如目标识别和理解为止,其典型流程如图8.12所示^[2]。

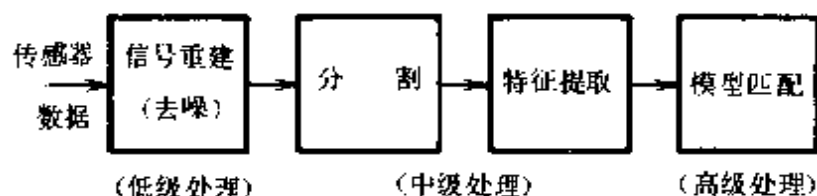


图 8.12 传统视觉系统处理流程图

虽然上述系统在不同的应用领域取得不同程度的成功,但在传感器数据噪声很大时却无能为力,因为带噪声的传感器数据将降低所计算的特征如分割标识、边界特征及序列处理中的惯性特征等质量,且这些噪声将不可避免的带来二次误差,同时,随着处理层次的上升,这些误差积累越来越多,使得处理精度越来越难保证,甚至得不到期望值。传统视觉系统的这种处理上的问题其主要原因在于高层处理阶段和低层处理阶段之间缺乏一优化反馈,例如,确定物体的比例和旋转等参数的算法主要依靠通过匹配边界的不连续性和计算空间矩(形心、惯性矩等)来实现。在低信噪比情况下,由于相关噪声过程的统计特性与物体成象的统计特性不一样,上述算法所确定的结果就会很不可靠,会导致大量的伪结果的产生,且传统视觉处理系统会把由上述所求得的低层特征中与噪声有关的伪结果代入后续更高层处理过程中,导致了第二代伪结果的产生。出现上述现象,主要是由于系统中与数据有关的反馈技术缺乏明确的一致性和优化性。该困难的产生在于传统的视觉系统中高层处理采用句法处理,如果采用反馈,就必须在高层次的句法处理和低层次的数值处理之间采取一些协调措施,但是,句法的表达又不能很方便地修改。例如,非线性句法滤波常常被用来清除有噪声的边界和标识,因而一旦出现误差,就要求能改变非线性句法滤波模式,然而,句法表达的不灵活性限制了此改变,造成非线性句法滤波在有噪声的识别环境中效果不

理想。

采用遗传算法所形成的视觉系统可以消除上述传统视觉系统所带来的限制。整个视觉系统(包括物体的识别和比例、方向、平移等参



图 8.13 采用遗传算法的视觉系统

数的估计),先形成一个简单的可最大化的目标函数,然后,通过作用于该目标函数上的大规模随机优化处理,所有的图像处理操作均自然地、自动地和协调地实现。该过程包括协同优化低层和高层变量之间的相互作用、满足模式的限制及高层模型信息的反馈等,并且该目标函数可直接对传感器原始数据进行计算而无需计算一些中间特征,即该方法避免了基于数据特征计算所带来的许多问题。整个系统框图如图8.13所示。

8.2.2 数学模型

1. 传感器数据模型

为了讨论问题的方便,我们把环境限制为在有限光学条件下二维物体的图像,因而,传感器数据可被模型化为整数值的二维矩阵,且该数据服从泊松(Poisson)分布,同时在某物体的子部分上的泊松点过程的密度为一常量。该矩阵在位置 (x, y) 的元素表示为 $s(x, y)$,整个传感器矩阵表示为 s ,物体背景也假定为一常量密度分布 λ_0 。在此,我们要强调的是,上述假定只是为了使得求解问题具体化,而与遗传算法进行识别无关。虽然非泊松分布情况在此并没有讨论,但用遗传算法仍然可以有效地解决非泊松分布时的图像情况。

2. 物体模型

假定所摄取的物体是由有限数量的多边形子部分所组成,这些物体的子部分在用分布图形表示时被表示为节点(见图8.14)。索引号为 m 的物体模型在用分布图形表示时其节点的总数量定义为 T_m 。节点之间的链代表多边形子部分的接触关系。这种形式的模型

足以表达一系列诸如飞机、坦克、汽车、家俱等人造物体的平面图。

节点和链的分布可被用来存储有用的特定物体的信息,每个物体子部分的密度可被表示为具有对应的图形节点的分布。具体来说,模型 m 的多边形子部分 k 的密度表示为 λ_{mk} 。每个链用来表示相邻子部分的分布,子部分 i 与 j 之间链的分布定义为 ρ_{ij} , ρ_{ij} 是一个子部分 i 相对于子部分 j 空间位置元素编码的向量。对于运动情况,链的分布还代表了子部分 i 相对于子部分 j 的方向角度参数,在通常情况下,这些角度参数限制于按一些特定步长变化。

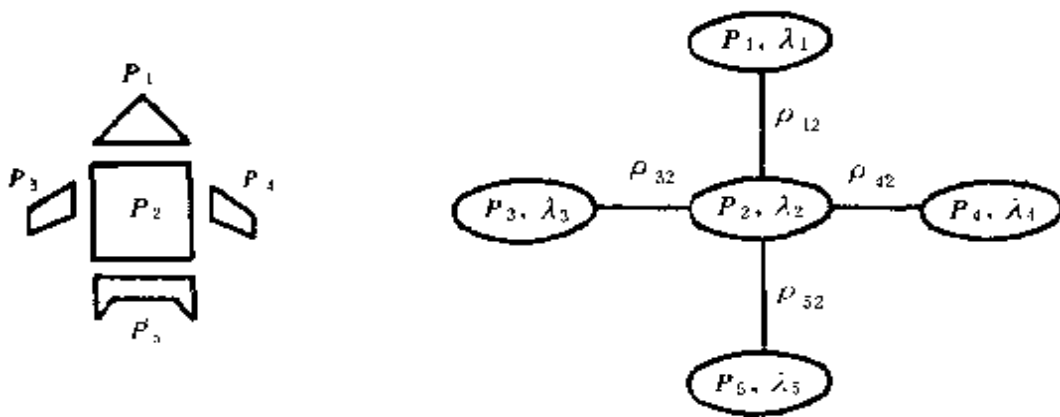


图 8.14 物体模型表示

3. 模型库

分布图形所表达的不同物体的集合组成了智能图像系统的模型库。现在我们假定该物体模型库包括了所有可能被摄取的物体(当然这不太可能),这些物体可能在任意二维位置、以任意方向和比例出现于图像空间,那么,我们的工作就是确定这些物体的类型和参数。

8.2.3 目标函数形成

所感兴趣的物体识别的主要参数包括二维位置坐标 γ , 方向 θ , 比例 γ 和物体的分类标号 m 。因而,整个图像识别问题的一个可能解可以表示为如下向量:

$$G_i = [r_i, \theta_i, \gamma_i, m_i] \quad (8.16)$$

它表示一个物体模型为 m_i 、其二维空间位置为 r_i 、相对于参考轴的

角度 θ_i 、比例为 γ_i 的情况。目标函数(表示为 $\mu(G_i|s)$)映射为一个可能解的得益值,该得益值代表了可能解 G_i 与传感器数据 s 之间的匹配度。从优化算法的角度看,可能解越逼近正确解,目标函数所返回的值越大。

这样,一个可能解的总得益表示为 $\mu(G_i|s)$,它是一个两参数的函数,一个参数是“覆盖得益”(coverage merit),另一个参数是“适应得益”(the merit of fit)。其中覆盖得益衡量了用该可能解解释被摄物体空间延伸的程度,该测量基于估计可能解与传感器数据之间失配的像素点的数量;适应得益衡量了由该可能解所指定的模型适应传感器数据的程度。对于大多数物体,覆盖得益参数对物体的位置和比例变化敏感,而对其它参数变化不敏感;适应得益参数对物体的模型、方向、位置等变化敏感,而对比例参数变化不敏感。考虑到上述两个参数相对于所有求解图像问题的参数的敏感程度,总的得益函数 $\mu(G_i|s)$ 可用下式表示:

$$\mu(G_i|s) = \mu_c(G_i|s) + U(\mu_c(G_i|s) - t) \times \mu_c(G_i|s) \mu_f(G_i|s) \quad (8.17)$$

这里, $\mu_c(G_i|s)$ 表示覆盖得益, $\mu_f(G_i|s)$ 表示适应得益。在式(8.17)中, $U(\mu_c(G_i|s) - t)$ 是一个单位阶跃函数,它说明仅当覆盖得益大于某门限 t 时,高敏感度的适应得益才影响整个得益值。目标函数的这种设计方法是确保仅当空间参数如位置和比例充分接近最终解时,目标识别和方向的参数才起作用。详细的覆盖得益和适应得益可表述如下:

(1) 覆盖得益

每一个可能解,都对应于一个包含有特定位置、方向和比例参数的假定的物体模型,它把图像空间划分成两个集合:一个集合对应于内部(前景)像素;另一个集合对应于外部(背景)像素。该划分示意图如图8.15。如果把内部和外部像素点的集合分别表示为 M 和 M_c ,那么一个可能解 $G = [r, \theta, \gamma, m]$ 的覆盖得益为:

$$\mu_c(G_i|s) = (1 + \sum_{(x,y) \in M_c} C_f(x,y) + \sum_{(x,y) \in M} C_b(x,y))^{-1} \quad (8.18)$$

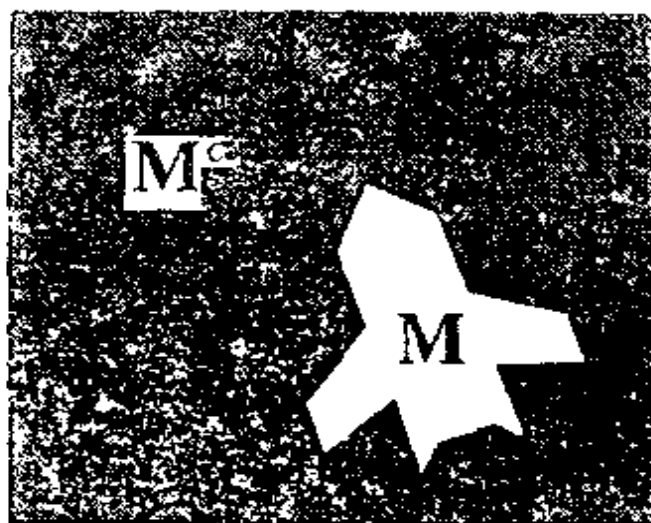


图 8.15 图像空间的划分

这里, $C_f(x, y)$ 表示 (x, y) 处的像素点在前景处理中的关系而 $C_b(x, y)$ 表示该像素点在背景处理中的关系。 $C_f(x, y)$ 和 $C_b(x, y)$ 在 $[0, 1]$ 间取值, 且该值越接近于 1 表示该分类的置信度越大。因此, 当大量的背景像素点由可能解错误地指定为属于物体区域时, $\sum_{(x,y) \in M} C_b(x, y)$ 就增大。类似地, 当大量的前景像素点由可能解错误地指定为属于背景区域时, $\sum_{(x,y) \in M_c} C_f(x, y)$ 就增大。这也就是说, 当可能解 G 所错误假定的像素点总数越小, 覆盖得益的值越接近于 1。 $C_f(x, y)$ 和 $C_b(x, y)$ 的值取决于传感器数据 $s(x, y)$ 及前景和背景过程的密度模型。函数的详细表达式可通过定义一系列贝叶斯假设测试来分类像素点, 使用联合错误分类概率的切尔诺夫界限 (Chernoff bounds) 作为分类置信度测量。具体的表达式可参阅有关文献[4]。

(2) 适应得益

如前所述, 物体模型可以表示为由多边形元素所组成的分布图形。对于每一多边形子部分, 就可以计算它相对于传感器数据的适应概率。这样, 整个模型的适应得益就可以通过对所组成的多边形的适

应概率实施加权几何平均值得到。其表达式如下：

$$\mu_f(G|s) = \left(\prod_{k=1}^{\Gamma(m)} \{P(\text{polygon } k \in H_f | s)\}^{\alpha_k} \right)^{1/\sum_k \alpha_k} \quad (8.19)$$

这里, $\Gamma(m)$ 是模型 m 的多边形子部分总数。在式(8.19)中, 因子 α_k 是加权因子, 用来强调物体的子部分区别于其它部分的实用程度。

这样, 从式(8.17), (8.18)和(8.19)我们可以得到总目标函数。物体的识别和位置、旋转、比例等参数的估计可以通过最大化该目标函数得到。

8.2.4 随机全局优化方法

当形成目标函数后, 最大化该目标函数就可得到所求结果。通过对该目标函数分析我们可以发现, 该函数除了在围绕其真实解的一个很小区域外在绝大部分均是非常平坦的, 在围绕真实解的这个很小区域里, 沿着解向量的每一维均有若干个尖锐的局部极大值, 而正确解对应于该区域内的最大值。由于该目标函数的平坦部分仍存在着大量的极小值, 基于梯度的方法就不适于求解这类问题。模拟退火近几年被广泛地用于基于目标函数的全局优化, 然而, 由于该方法潜在的串行性、解空间的随机搜索性等增加了实现的时空复杂度。基于上述原因, 我们采用了遗传算法。其优点为: 1) 并行搜索策略。2) 基于良好的部分解的方法。

遗传算法是一种随机优化算法, 该算法对目标函数的限制较少, 而仅仅需要评价每一个候选解的函数值。遗传算法特别适用于对于问题存在固定的部分解的情况, 一个好的部分解可能对应于我们求解问题中一个或多个良好的估计值, 遗传算法能够以某种方式联合各部分解以带来更好的总体结果。

在遗传算法中, 每一候选解表示为一位串, 遗传算法通过其三个基本操作: 选择、交叉和变异来完成对该位串的修改以最大化目标函数。然而, 在实际中, 该算法要花费大量时间才能收敛到最大值, 如对

某一特定求解问题引入专门知识以帮助收敛往往能取得意想不到的效果。因此,有必要对“标准”遗传算法作一些修改。

对于交叉操作我们通过引入所求解的特定问题的知识来改进,具体如下:当进行交叉操作的两个候选解的目标函数值低于一门限时,实施标准的单点交叉操作。当进行交叉操作的两个候选解之中其任一目标函数值大于某一门限时,交叉参数从该两候选解的位置、旋转、比例参数的平均值选择,再对该两候选解的目标函数值加权。有效模型参数随机地从候选解中选择。这样形成的新候选解加到候选解池中以代替目标函数值低于该新候选解的其他候选解。但有一点需要说明的是,仅当两候选解中至少一个有点接近最优解时才引入改进的交叉操作。基于这个原因,也即两候选解中至少一个的目标函数值大于一固定门限时才引入改进交叉操作。

变异操作是用来随机扰动可能解以跳出局部优解。类似于交叉操作,变异操作也引入了特定问题的知识。例如,在本问题中,可能解的局部最大值趋向于出现在很接近于正确的位置附近,但该可能解的旋转、比例、模型参数也许在距最优解相差较远的地方。此外,在靠近最优解的地方也有若干个局部极大值。基于这些原因,变异操作设计如下:池中一小部分可能解保持不变以保证不丢失最佳可能解;剩下一小部分通过在其解的参数上(位置、方向、比例)加上一小的常数实现轻微变化,该操作对应于已靠近最优解附近的局部极值。最后的可能解的变异按如下步骤实现:小的随机扰动值加到位置参数,大范围的随机扰动加到旋转、比例和模型证实参数上。变异后的可能解代替以前可能解以跳出局部优解。一旦可能解已收敛到接近最优解,则逐步减少变异操作概率。

8.2.5 实验结果

该算法已在包含有40个32位 INMOS T800 Transputers 的并行结构上实现。位串编码如图8.16,结果见图8.17及8.18。图8.17显示了组成模型库的4个物体模型。每个模型均由泊松(Poisson)随机数

据产生,以白轮廓线显示于屏幕上。4个模型分别命名为坦克、飞机、汽车和火箭。

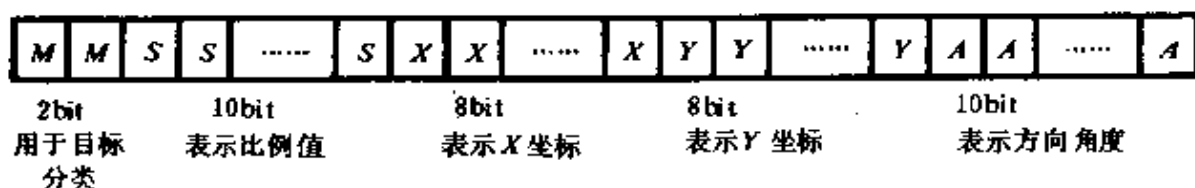


图 8.16 位串编码

图8.18显示了坦克数据集的遗传进化过程。在这些照片里,每个可能解以白轮廓显示其模型、位置、比例和旋转参数。其中第一行第二列的照片显示的是初始候选解,在本实验中,个体数为128。初始成员设定为绕一个圆均等分布,比例和旋转参数设定为绕着距可能解的等间隔分布。当然,这种初始化不是唯一的。在第3代(第1行第3

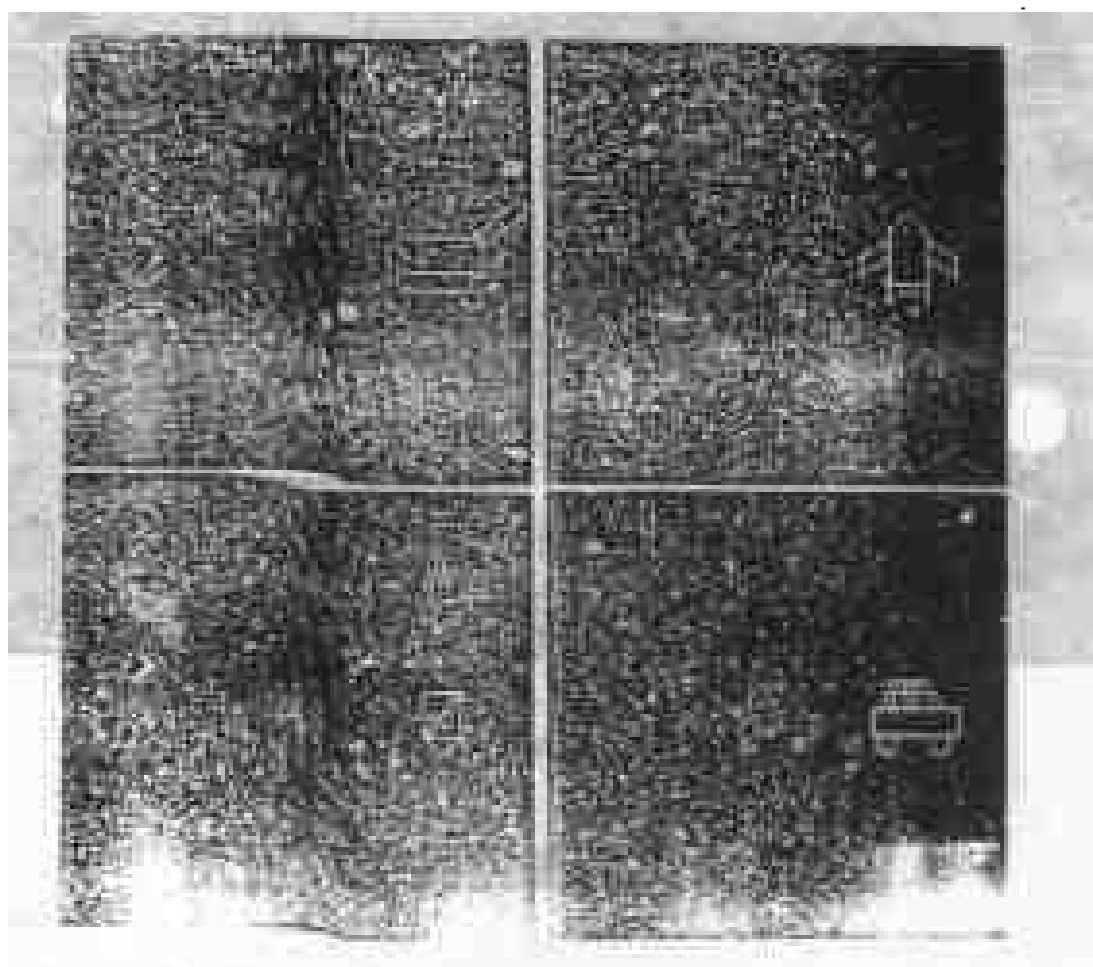


图 8.17 模型库

列),可能解开始聚集在坦克的位置附近。一代以后(第1行第4列),所有可能解围绕着物体收敛到一个小的区域,这是因为使用了改进的交叉操作导致了快速收敛。在第7代(第2行第1列)所有可能解临时收敛到汽车模型,在下面几代中,即用变异操作跳出局部优解。在第10代(第3行第1列),可能解已经逼近正确位置、模型和比例参数。在第17代(第3行第2列),可能解已经收敛到正确解。最后一行照片显示了从第18代至第20代的情况,因为所有可能解均为较为理想,所以照片上只是显示了一个轮廓。

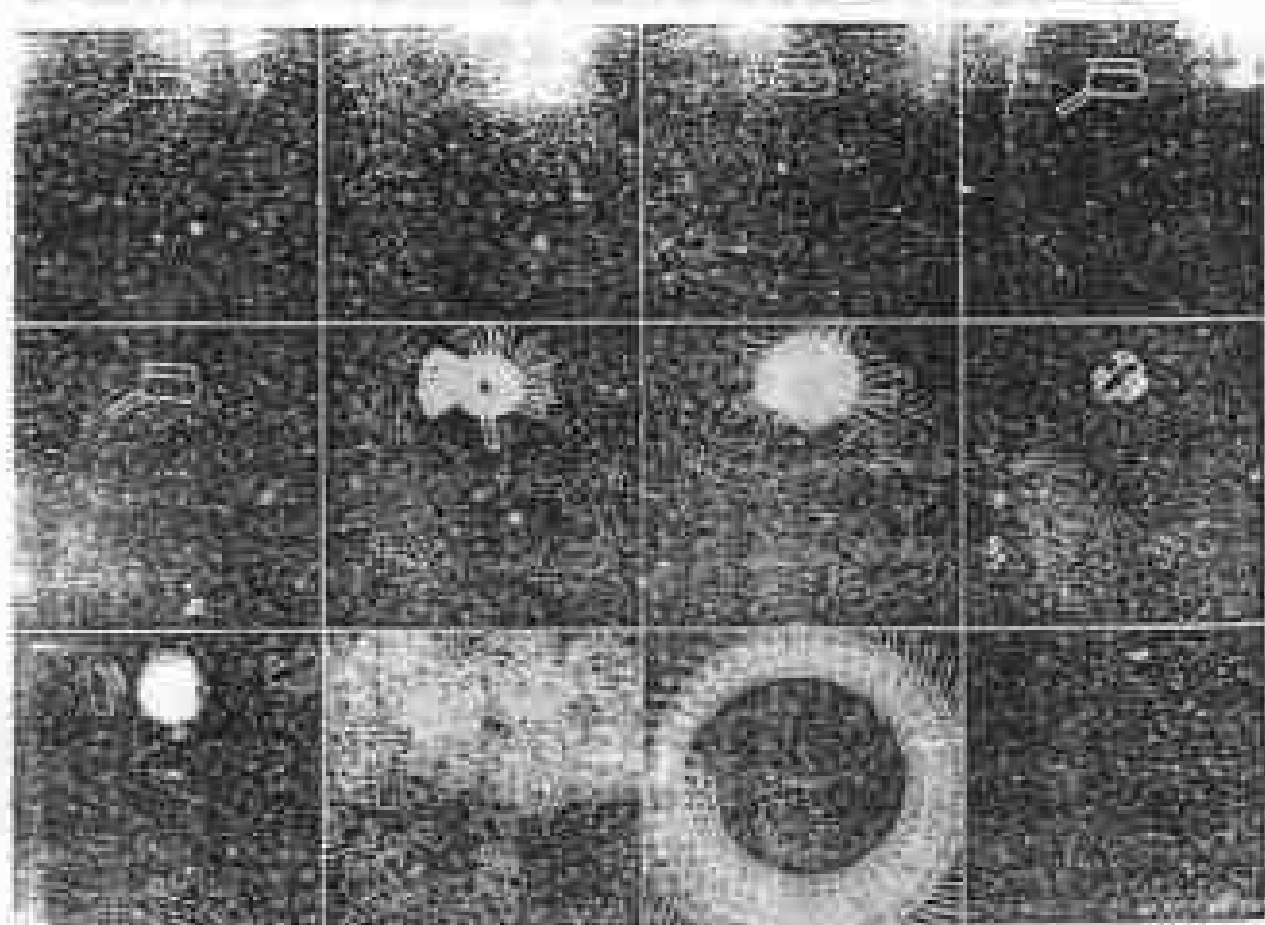


图 8.18 遗传算法优化过程

8.3 遗传算法在控制中的应用

遗传算法作为优化方法用于控制中是十分有意义的。目前,有

关这方面的应用研究已不少成功事例^[5],其中,大多数都采用了以下两种方法:

(1) 把控制量或控制规则直接编码为染色体进行遗传操作。这里又可分两种形式:一种是对操作量序列进行编码;另一种是先对状态空间进行适当划分,然后让各子空间对应基因座,而操作参数对应基因值。

(2) 把控制参数编码为染色体进行遗传操作。比如,在基于神经网络的控制中,可用 GA 代替神经网络的学习方法来训练权重,又比如在模糊控制中,可用 GA 来调整隶属函数。

下面我们将详细介绍这两种方法。

8.3.1 操作序列的最优化^[6]

让原子反应堆停止工作时,有可能产生对原子反应堆有害的钚(S_m)。为了尽可能地减少钚的产生,可通过对中子束的优化控制来实现。这一控制可用下面的微分方程来描述:

$$\begin{aligned} \dot{P}_t &= Y_p \cdot \sum f \cdot \Phi_t - \lambda_p P_t \\ \dot{S}_t &= Y_p \cdot P_t - \sigma_s \cdot \Phi_t \cdot S_t \end{aligned} \quad (8.20)$$

式中: P_t 为时刻 t 中 $^{149}\text{P}_m$ (钚) 的浓度;

S_t 为时刻 t 中 $^{149}\text{S}_m$ (钚) 的浓度;

Φ_t 为时刻 t 中的中子束;

Y_p 为 $^{149}\text{P}_m$ 的生成比例;

λ_p 为 $^{149}\text{P}_m$ 的衰减常数;

σ_s 为 $^{149}\text{S}_m$ 的中子吸收的断面积;

$\sum f$ 为核裂变的宏断面积。

假如原子反应堆从 t_0 开始停止操作,并要求在 t_c 时刻完全停止工作。这一过程中,中子束强度也从 $\Phi_0 = \Phi_{\max}(t_0)$ 经控制达到 $\Phi_t = 0 (t > t_c)$ 。在对 Φ_t 的控制过程中,要保持 S_m 的浓度 $S_t (0 < t < t_c)$ 低于某个阈值,而在原子反应堆完全停止后 $S_t (t > t_c)$ 要尽可能地减少。这里我

们采用遗传算法来进行控制。

(1) 编码。将规定的控制时间 t_c 分割成 N 个区间。各区间的 Φ_i 值离散化后用整数表示。这样,染色体就编码为 N 个整数组成的集。

(2) 适应度计算。用染色体所规定的操作序列进行原子反应堆的停止操作,并在规定的时间内测定 S_m 的浓度值作为适应度。在仿真实验中,控制时间为80小时,适应度评估时间为从控制开始100小时后。

(3) 遗传操作:群体大小为20,采用简单的 GA 方法。但在进化中,每代约有20 %的好个体原封不动遗传到下一代。余下来的个体中,60 %的个体按适应度比例准则挑选出来直接复制到下代,10 %的个体进行多点交叉,还有10 %的个体进行单座变异。

这个例子由于只有一个操作量,所以比较简单,但是,若对编码作些改进,该方法可扩展到多个操作量同时控制的问题中。由于这种编码要对操作序列进行分割,若分割得很细(微秒级分割),则会使染色体很长,从而增加计算量。这是需要进一步改进的。

8.3.2 倒立摆控制^[7]

倒立摆控制的概念如图8.19所示。在时间 t ,系统的状态可用如下的四个状态来描述:

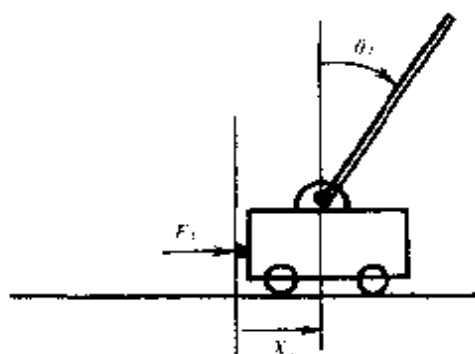


图 8.19 倒立摆

X_t : 小车的速度

θ_t : 摆的倾斜度

x_t : 小车离开中央的距离

$\dot{\theta}_t$: 摆的角速度

上述的控制量即操作参数是加在小车上的作用力 F_t 。学习系统将决定在各时刻加在小车上的力的作用方向(左还是右),作用力的大小是固定不变的。

倒立摆的运动方程如下:

$$\ddot{\theta}_t = \frac{g \sin \theta_t + \cos \theta_t \left[\frac{-F_t - m_p l \ddot{\theta}_t \sin \theta_t}{m_c + m_p} \right]}{l \left[\frac{4}{3} - \frac{m_p \cos^2 \theta_t}{m_c + m_p} \right]} \quad (8.21)$$

$$\ddot{X}_t = \frac{F_t + m_p [\ddot{\theta}_t^2 \sin \theta_t - \ddot{\theta}_t \cos \theta_t]}{m_c + m_p}$$

$g = 9.8 \text{ m/s}^2$ (重力加速度)

$m_c = 1.0 \text{ kg}$ (小车质量)

$m_p = 0.1 \text{ kg}$ (摆质量)

$l = 0.5 \text{ m}$ (摆长度)

基于上述方程,用 Euler 方法,以时间间隔 $\tau = 0.02 \text{ s}$ 计算出各时刻系统的状态。

在基于遗传算法的仿真控制中,若摆左右倾斜 12° 以上,以及小车的位置偏离中心线 2.4 m 以上时视为失败。

(1) 编码。这里,我们采用先对状态空间划分,然后各子空间作为基因座,而操作参数作为基因值的编码方法。

为简化问题,我们忽略状态变量 X ,对余下的3个状态变量作如下划分:对 X 和 θ 作三划分,即有

$$\dot{X}: -\infty \sim -0.5, -0.5 \sim 0.5, 0.5 \sim \infty (\text{m/s})$$

$$\dot{\theta}: -\infty \sim -50, -50 \sim 50, 50 \sim \infty (^\circ/\text{s})$$

对 θ 作六划分,即有

$$\theta: -12 \sim -6, -6 \sim -1, -1 \sim 0, 0 \sim 1, 1 \sim 6, 6 \sim 12 (^\circ)$$

这样一来,整个状态空间被划分成54个领域($3 \times 3 \times 6$),也就是说,染色体有54个基因座,每个基因座对应的基因值不是“左”就是“右”。在计算机上仿真时,右对应1,左对应0。所以染色体就是由54位组成的二进制集。

(2) 适应度计算。用染色体中所表示的操作量来控制倒立摆(即或左或右地对倒立摆加力)从控制开始到倒力摆倒下为止的这段时间作为适应度值。

(3) 遗传操作。群体大小为300,采用简单的GA方法,但略微作了一些操作变动。在进化中,超过群体平均适应度的个体被复制到下一代。但是,1)如果超过群体平均适应度的个体数小于群体中个体总数的10%,那么从群体中按适应度高低从上到下挑选10%个体复制。2)如果超过群体平均适应度的个体数在群体中的比例大于60%,则按适应度大小,从上到下保留60%的个体。3)假如(2)的状态连续3代出现,则仅保留其中20%的个体。对个体进行上述的复制或保留后,对余下的个体进行一点交叉。此外,对所有的子孙按0.01的概率进行变异。

显然,上述学习控制多状态空间的划分程度影响,所以适当地划分状态空间是十分重要的。作为改进,可以把状态空间的划分的方法也包含在编码中,以获得优化的状态空间分割。

8.4 调度问题

所谓调度(scheduling),就是为了实现某一目的而对共同使用的资源实行时间分配。例如,车间作业调度问题(JSSP; Job Shop Scheduling Problem)就是为了处理多项不同的事务而如何分配作为共同资源的机械设备,并使总的作业时间最少的问题。一般认为,这种JSSP是NP完全问题中最困难的问题之一,即使JSSP中比较简单的流动车间调度问题(FSSP; Flow Shop Scheduling Problem),也是与城市距离不对称情况下的货郎担问题(TSP; Travelling Salesman Problem)难度相当的同一类型的问题。

以往,JSSP的求解方法主要是分枝界限法(BAB; Branch And Bound)。采用标准检查程序(Benchmark)来评价解的结果。近来,对于上述这类具有复杂解空间的问题有了一些效果比较好的新求解方

法,例如,神经网络、模拟退火、遗传算法等等。其中的遗传算法(GA)由于具有处理问题的柔软性和并行处理的能力,使得它得到了广泛的应用。

8.4.1 车间作业调度问题

考虑 n 项事务,在 m 台设备上处理的问题。其中,假设每台设备不能同时处理两项以上的事务。反过来,各种事务也不能在两台以上的设备上同时处理完毕。我们把每台设备上各种事务的处理称为作业,把每项事务所需要的设备使用顺序称为技术顺序。各作业的处理时间是预先给定的。

一般地, $n \times m$ 调度问题,可以有以下三种分类方法^[9]:

(1) 根据设备环境可分为开放式车间(O):不特别给定某一技术顺序;流动车间(F):给定技术顺序但不能预知事务;作业车间(J):给定技术顺序和事务但每项事务都不相同,等等。

(2) 根据事务特点可分为允许作业中断(P_{pmtu}),作业处理时间全部相等(Pu),处理时间不限制(G),等等。

(3) 根据某一最佳标准来分:总作业时间(C_{max})最小,总延迟(L_{max})最小,等等。

如果采用上述一些分类方法,则这里所考虑的“总作业时间最小的一般 JSSP”可表达为 $J|G|C_{max}$ 类型的问题。

8.4.2 两种解法

用 GA 求解给定的问题时,最重要的两点是解的表示与交叉。因此,用 GA 求解 JSSP 可以考虑如下三种方法。

(1) 对位序列编码,使用简单 GA 求解。

(2) 对含有顺序的问题,用顺序列上的交叉求解。

(3) 利用问题特性进行求解。

上述第(2)种方法适用于货郎担问题,流动车间型的调度问题^[8]。因此,这里只讨论第(1)、(3)两种方法。

1. 用简单 GA 的求解方法

用简单 GA 求解 JSSP,除了要巧妙地运用 GA,调整各种参数以外,还要着重考虑以下三点:(1) 解的表示方法;(2) 非解到解的变换方法;(3) 保存优者(elitism)的策略,下面分别予以讨论。

(1) 解的表示方法。

如果决定了各设备上事务的序号,则可以唯一地确定 JSSP 的解,即可以用每台设备执行事务的集合来表示解。解的表示方法有符号表示和位表示两种。由于简单 GA 用的是位表示,所以这里也采用位表示方法。符号表示法参见图8.20(b)。

JSSP 的位表示方法也可以有几种。最简单的方法是对解中所使用的符号(事务序号等)分别赋以适当的位序列,而不考虑各个符号之间的相互关系。但是,这种位序列中各个位并没有特别的意义,不同位表示之间的类似性与不同解之间的类似性之间没有什么相关性。所以这种表示方法不理想。

这里,我们采用注重事务对的方法。所谓事务对,也就是考虑了相互关系的一对事务。由于考虑了事务之间的相互关系。因而表示事务对的位序列中各个位都有明确的意义。图8.20是一个具体的例子,它考虑了一个 6×6 问题所具有的所有15种事务对。我们注意图8.20(c)的第一行,它考虑了 job1 和 job2 这对事务 job1 比 job2 先处理,图中用符号“ $<$ ”表示。图中位序列是这样产生的,序列长度由设备数目确定,这里为6;序列中各位的顺序依据 job1 所用的设备序号来进行,这个顺序由技术顺序确定(由图8.20(d)的第一行看出,job1 的技术顺序为:设备3 \rightarrow 设备1 \rightarrow 设备2 \rightarrow ...);按照这样的顺序,从图8.20(b)中分别查看每台设备上事务的顺序,job1 优先于 job2 的,位顺序中相应的位置为1,否则置为0,这样,6台设备就产生了一个6位的位序列,如图8.20(c)中第1行所示。其它事务对所对应原位序列可类似地产生。全部列于图8.20(c)。图中,对于每一个位序列,相邻两位若是相同数字(0或者1),则表示事务对之间没有相互超越现象,即不改变事务对中两个事务的处理顺序。图8.20(c)的位表示中,相邻位为


```

machine1:      111      44444333333333 6666666666222222222555
machine2: 2222222244444666111111555  3
machine3: 333331  222255555555544444      6
machine4:      3333      666      444111111      22225
machine5:                222222222  55555333333344444446666111111
machine6:                33333333  6666666662222222225555111444444444

```

(a) 最优调度 (总作业时间)

| | |
|-----------------------|------------------------|
| machine1: 1 4 3 6 2 5 | job1<job2: 1 1 0 1 0 0 |
| machine2: 2 4 6 1 5 3 | job1<job3: 0 1 1 0 0 0 |
| machine3: 3 1 2 5 4 6 | job1<job4: 1 1 0 0 1 0 |
| machine4: 3 6 4 1 2 5 | job1<job5: 1 1 1 1 0 0 |
| machine5: 2 5 3 4 6 1 | job1<job6: 1 1 0 0 0 0 |
| machine6: 3 6 2 5 1 4 | job2<job3: 1 0 1 0 0 0 |

(b) 符号表示

| | |
|-------------------|------------------------|
| job1: 3 1 2 4 6 5 | job2<job4: 1 1 1 1 0 0 |
| job2: 2 3 5 6 1 4 | job2<job5: 1 1 1 1 1 1 |
| job3: 3 4 6 1 2 5 | job2<job6: 1 1 1 0 0 0 |
| job4: 2 1 3 4 5 6 | job3<job4: 1 1 1 0 0 1 |
| job5: 3 2 5 6 1 4 | job3<job5: 1 1 1 1 0 0 |
| job6: 2 4 6 1 5 3 | job3<job6: 1 1 1 1 0 1 |
| | job4<job5: 1 1 0 1 0 0 |
| | job4<job6: 1 1 1 0 1 0 |
| | job5<job6: 1 0 1 0 0 0 |

(c) 技术顺序

(d) 位表示

图 8.20 JSSP 解的表示方法

相同数字的情况比较多,因而事务间的超越现象就比较少。没有任何超越现象的处理方法就是流水线处理。流水线处理可能成为一种最佳调度。只要它具有一些比较好的初始条件,即初始条件的位序列不是随机产生,而是要通过调整相同的相邻数字的概率产生。

(2) 非解到解的变换法。

从初始条件得到的位序列或从 GA 操作得到的位序列,一般并不是解的位序列表示。例如,对于 6×6 问题,解的位序列长度应该是 90,即图 8.20(c) 中 15 个 6 位序列的长度之和,所以基于位序列表示的候选解的数目是 $2^{90} \approx 10^{27}$,而基于图 8.20(b) 所示的符号表示的候选解的数目是 $(6!)^5 \approx 10^{17}$ 。这个数比 10^{27} 小了许多。而且,有一部分符号序列,并不表示解。所以实际解的数目比 10^{17} 还要少。这就是说,用位表示方法表示的 10^{27} 种位序列中,有相当一部分根本就不是问题的解。这样,就有必要把那些非解的位序列 A 变换成解的位序列 C ,并且希望 C 尽可能地类似于 A (但决不能完全相同)。这里的类似性是指位序列中反转位的数目(近似于海明距离)比较少。下面我们讨论海明距离尽可能小的非解到解的变换法: LH/GH 算法^[11]。LH/GH 算法分为两个阶段。第一阶段为局部调和(LH)算法,它把非解序列 A 变换成符号表示的序列 B 。第二阶段为全局调和(GH)算法,它把符号表示的序列 B 变换成解的位序列 C 。

LH 算法是各设备独立地进行,即各个设备都分别决定其所承担事务的处理顺序。非解序列 A 包含了与事务处理顺序有关的信息。但这些信息可能存在着某些不协调,也就是事务对的处理顺序发生了矛盾。LH 算法就是要消除这种不协调性,并保证海明距离最小。第一阶段处理结束以后,各设备上的事务处理顺序也就确定了,图 8.21 表明了设备内不协调性逐渐消除的过程。从而得到了解的符号表示的序列。

为了能用 GA 对解进行优化,还必须用 GH 算法把符号表示的序列 B 变换为解的位序列 C 。事实上,当调度程序在逐渐生成调度的过程中陷入不能调度的状态时,也就是当出现了某台设备同时处理

| | 和 | | 和 |
|-------------------|-------------|-------------------|-------------|
| job1: * | 0 0 1 1 0 2 | job1: * | 0 0 1 1 0 2 |
| job2: 1 * | 0 0 1 1 3 | job2: 1 * | 0 1 1 1 4 |
| job3: 1 1 * | 1 1 0 4 | job3: 1 1 * | 1 1 1 5 |
| job4: 0 1 0 * | 0 0 1 | job4: 0 0 0 * | 0 0 0 |
| job5: 0 0 0 1 * | 1 2 | job5: 0 0 0 1 * | 1 2 |
| job6: 1 0 0 1 0 * | 2 | job6: 1 0 0 1 0 * | 2 |

(a) 初始化

(c) 选择任务 2 以后

| | 和 | | 和 |
|-------------------|-------------|-------------------|-------------|
| job1: * | 0 0 1 1 0 2 | job1: * | 0 0 1 1 1 3 |
| job2: 1 * | 0 0 1 1 3 | job2: 1 * | 0 1 1 1 4 |
| job3: 1 1 * | 1 1 1 5 | job3: 1 1 * | 1 1 1 5 |
| job4: 0 1 0 * | 0 0 1 | job4: 0 0 0 * | 0 0 0 |
| job5: 0 0 0 1 * | 1 2 | job5: 0 0 0 1 * | 1 2 |
| job6: 1 0 0 1 0 * | 2 | job6: 0 0 0 1 0 * | 1 |

(b) 选择任务 3 以后

(d) 最终优先级

图 8.21 局部调和(LH)算法

两项以上的事务,或者某项事务在两台以上的设备同时处理时,启动 GH 算法。GH 算法可以消除不能调度的状态,但不能保证解获得最少反转位数。既要使获得位序列 C 中总反转位数最少,又要消除不能调度的状态,则是一个很困难的优化问题,往往难以解决。特别接近于 A 的位序列 C 并不一定是好序列,一味地追求之也并非上策。

(3) 优者保存策略。

若把非解的位序列 A 视为基因型(genotype),把解的位序列 C 视为表现型(phenotype),则前述的 LH/GH 算法就是基因型到表现型的变换过程。这个过程不是确定性的,因为变换过程中可以随机地选择某些分枝。非确定的 LH/GH 算法偶尔也产生一些优秀解,但不

能保证这些优秀解的可重复性。因此,可以考虑采用一种优秀解保存策略。这种策略仅仅把那些最优秀的表现型再置换成基因型(我们称之为强制作用(forcing)),使之遗传给下一代。采用这种策略能获得明显的优化效果,如图8.22所示。

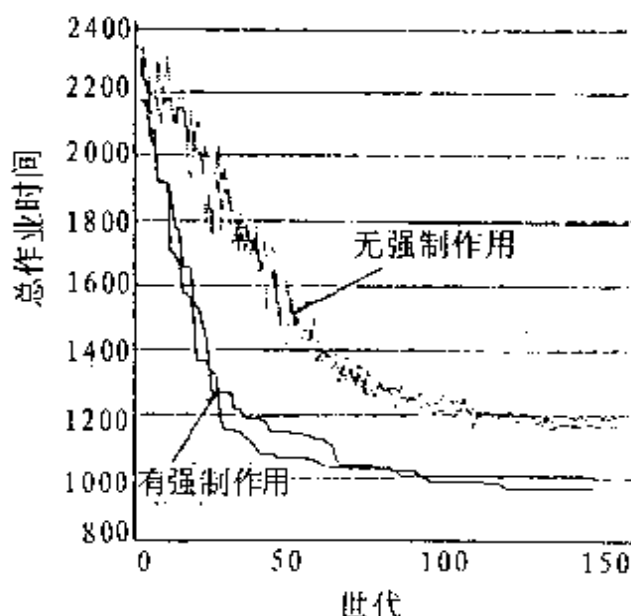


图 8.22 强制作用的效果

2. 利用问题特性的求解方法:GA/GT 法

(1) 一致交叉。

简单 GA 中的交叉是指通过对两个父代个体的位序列中的若干位进行交换操作,生成两个新的子代个体的位序列。若分别在两个位序列中随机地各选择一位进行交换,就是二点交叉。

一致交叉(uniform crossover)是对多点交叉的扩展^[12]。考虑与个体具有相同长度的一个随机位序列,并称之为屏蔽位序列。在屏蔽位序列中与1相对应的地方,交换遗传子。换句话说,对于第一代子代位序列中的每一位,如果对应的屏蔽位为0,则是从父代的一个序列中相应的位通过复制继承而来,如果为1,则是从父代的两个序列中相对应的位通过交叉继承而来。而且,使用同样的屏蔽位序列,通过两个父代序列的任务交换也可以得到子代序列。对于一致交叉,视其对屏蔽位序列设置方法的不同,可以形成各种各样的交叉。例如,某

屏蔽位序列以0为主,序列的开头和结尾都是0,序列中间偶尔有个别的1。在这种情况下的一致交叉就变成二点交叉。

(2) GT 算法。

利用 JSSP 的问题特性的 GA 求解方法,需要定义一种与一致交叉相当的交叉。在简单 GA 的情况下,不是取出二次调度的一部分进行再次组合,而是在参考两父代信息的同时,重新调度。

这里所用的调度算法是以 Giffler 和 Thompson 的活性调度法(GT 算法)为基础的。在下面的 GT 算法中,作业的最早完成时刻指的是对该作业进行最优先处理的情况下的完成时刻。

第一步,在未处理的作业中,取出最早完成时刻的最小作业 o^* (处理时间最短的作业)。

第二步,建立冲突集 C 。集合 C 中的元素由 o^* 以及与 o^* 在同一设备(设为 M_i)上作业、并重复 o^* 的处理过程的那些作业组成。

第三步,从 C 中选择一种作业,按照最早完成时刻进行处理。

对于所有作业,都重复以上处理,就可以得到一种新的调度。所得到的调度称为活性调度(active scheduling)。在第三步选择作业的时候,如果考虑所有的可能性,则能产生全活性调度,但全活性调度所具有的可能的调度数目是相当大的。

(3) GT 交叉。

基于上述 GT 算法,我们来考察一种称为 GT 交叉的交叉方法^[13]。GT 交叉是对基于简单 GA 的一致交叉朝着有效地利用问题特性方向的扩展。但是在这里屏蔽位序列不再是一维的位序列,而是以二维随机阵列 $H(h_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$ 取而代之,其中 h_{ij} 表示第 i 台设备上第 j 个作业。当父代阵列给定时,屏蔽阵列 H 就能够完成新调度(子代调度)的设计任务。GT 交叉就是在 GT 算法的第三步从 C 中选择作业时,决定参照哪个父代调度。此时,适合于参照的父代,可用 h_{ij} 的值替换之。因此,第3步应作如下变更:

第三步,下次要选的作业,取 M_i 上的第 j 个作业。在两个父代阵列中,指定与 h_{ij} 值相对应的一个。再根据指定的父代阵列,从 C 中

选择一个处理速度最快的作业,并按照最早完成时刻进行处理。

GT 交叉是按照这样的意图定义的,即子代调度尽量忠实地反映父代调度的处理顺序。实际上,在第三*步中,当两个父代调度完全相同时,所产生的子代调度与原调度是一致的。在一般情况下,是产生具有与 H 的元素中含0和1的比例相对应的父代特性的调度。

(4) 变异。

通过对第三*步作如下变更,可以产生变异:

第三**步,与第三*步同样的方法,在 C 中选择一个作业。但是,所指定的父代调度并不是最快处理的作业,而是选择处理速度第 n 快($n>1$)的作业,只不过 n 取较大值的概率较小。

例如,在两个父代调度完全相同的情况下,变异的结果是,仅在发生变异那一点的周围产生不同的调度。因此,变异是在 GT 交叉中增加邻近探索的方法。

以上讨论的是从两个原有的调度产生第一代新调度的过程。对于第二代调度,也与一致交叉一样,用同一个屏蔽阵列 H ,通过两个父代调度作业序列的任务交换而得到。通过这种方法,使父代调度的特性全部遗传下去。如果把上述的 GT 交叉与现有的淘汰法组合起来,就组成了利用问题特性的求解方法,即 GA/GT 法。

8.4.3 实验

1963年,Muth 和 Thompson 提出了三个标准检查程序^[10],其中 6×6 的问题规模较小,比较容易。其余两个是分别 10×10 或 (20×5) 和 20×20 的问题。这两个问题规模大一些,求解比较困难,其中 10×10 问题几乎在20年的时间内都未得到解决,它是说明 JSSP 的求解复杂性的一个很好的例子。 10×10 和 20×5 问题的实验结果如图8.23和图8.24所示。其中 GA 的参数为,集团的个体数为2000,遗传代数分别为100(对于 10×10 问题)和200(对于 20×5 问题),变异率为0.01。例如对于 10×10 问题,运行了300次,其中有5次获得了总作业时间为950的解,在 Sparc 工作站上用 C 语言编程,每次运行时大约

为10分钟左右。

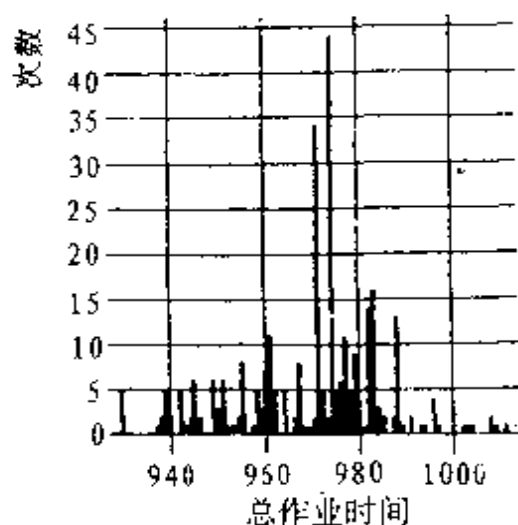


图 8.23 10×10问题解的分布

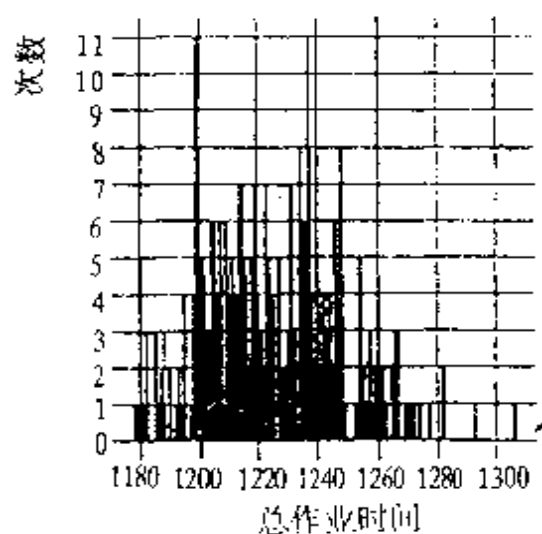


图 8.24 20×5问题解的分布

GA/GT 法还可适用于比较大规模的问题,例如20×20的问题。其实验结果如表8.1所示。表中左边第1列为4个问题的编号,其余数字为总作业时间。GA 的参数都与10×10问题相同,技术顺序和各作业的加工时间随机选定。对于每个问题运行10次。由表8.1看出,GA/GT 方法与用随机方法产生40万个活性调度比较而言,总作业时有明显的减少。

表 8.1 GA/GT 与随机方法的比较

| 20×20 No. | 随机 | | GA/GT | |
|--------------|--------|------|-------|------|
| | 平均 | 最好 | 平均 | 最好 |
| 1 | 1356.8 | 1126 | 975 | 965 |
| 2 | 1318.6 | 1104 | 950 | 928 |
| 3 | 1313.5 | 1107 | 962 | 943 |
| 4 | 1414.3 | 1202 | 1050 | 1033 |

8.5 硬件进化

所谓硬件进化(Evolvable HardWare,简称 EHW)^[14,15],就是为了在变化的运行环境中,硬件本身能够自动地改变结构以适应环境的变化,也就是使得具有某种功能的硬件能随着环境的变化而进化。对于这样的问题,传统的硬件设计方法是无能为力的。但是如果能够应用基于遗传算法的学习方法,并利用大规模可编程逻辑阵列的结构特点,通过选择阵列的功能,选择各逻辑阵列之间的连接方式,则有可能使硬件结构自动地进化,使之对未知环境也具有一定的适应性。

早期的可编程逻辑器件(Programmable Logic Device,简称 PLD)是 PLA(Programmable Logic Array)器件。但 PLA 的可编程能力是一次性的,即一旦通过烧断起连接作用的熔丝而确定了 PLA 的逻辑功能以后,则不能更改,这使它的应用受到了很大的限制。近年来,PLD 的规模(单片所含晶体管数目)和种类发展很快,已达到2万个逻辑门,速度也已达15纳秒左右,有可能构成高速的硬件系统。虽然不同的生产厂家的 PLD 具有不同的结构,但有两点是共同的,一是器件中包含逻辑宏单元,二是各逻辑宏单元之间的连接结构。PLD 功能,可通过指定某特定的位序列加以确定,即通过对位序的“编程”,形成任意的硬件单元功能。

逻辑宏单元是一个通用的逻辑电路,通过指定特定的位序列形成任意的组合电路。当有多个逻辑宏单元时,需要有一个连接它的连线机构。这种连线机构一般是一个2维的熔丝阵列。通过相应操作序列以指定熔丝阵列模式,实现各宏单元之间的不同连接,进而可实现任意的硬件电路。

FPGA(Field Programmable Gate Array)是当前 PLD 中最流行的一种。它由可选择多种功能的逻辑单元和决定 FPGA 整体功能的各单元之间的相互连接构成。逻辑单元可以包含与门、或门、触发器

等基本逻辑单元。通过它们的组合,又可以形成各种功能的逻辑单元。逻辑单元的描述采用特定的位序列。逻辑单元之间的相互连接也通过指定位序列来实现。这种指定位序列的过程相当于编程。而位序列本身也可称为位结构。

EHW 的基本考虑方法是,通过遗传算法的学习,在 FPGA 中寻找一种位结构,并把位结构看作染色体,使之能实现我们所希望的适应于环境的硬件功能。在本节中,我们首先讨论硬件进化的一些特点,然后介绍硬件进化的学习方法,最后用 GAL (Generic Array Logic) 电路给出两个实例,其中一个是组合电路,另一个是时序电路。

8.5.1 硬件进化的特点

硬件进化主要有两个特点,一是高速的运行速度,即可用有限状态机代替微处理器,从而实现实时控制,二是有较好的自适应性和容错性,即当环境发生变化或系统发生故障时能自动重新组织硬件结构。下面我们分别予以讨论。

1. 运行速度

在大多数自适应学习系统中,都是用符号网络或神经网络描述一些规则(例如分类规则等),并用来表示自适应的结果,这些系统基本上是软件系统。因此,当这些系统应用于实际问题时,难以实现高速的实时处理。本节将要讨论的硬件进化,由于采用了自适应的硬件结构,所以能实现高速处理。例如用 GAL16V8 芯片所设计的组合电路,运行时间不超过10纳秒,时序电路能够适应高达63MHz 的输入时钟。

2. 用有限状态机实现实时控制

在有限状态机中,主要是通过状态转移实现控制的,因此必须采用某种方法来表示其内部状态。如果用神经网络来表示内部状态,就难以实现实时控制,因为神经网络本身结构不能实现状态保持。要想实现状态保持,就必须增加反馈,这就增加了网络的复杂度,对实时

控制不利^[16]。与神经网络相比,EHW 中硬件单元本身就是状态机(例如触发器)因此很容易表示内部状态。

在 EHW 上实现的有限状态机比软件实现明显地快了许多,因此有可能实现具有实时要求的控制。我们以机器人控制中的归类结构(Subsumption Architecture)简称 SSA^[17]为例来说明这个问题。并与用软件设计的遗传编程方法进行比较。

SSA 是设计智能机器人的一种新方法,它采用若干个能实现关键动作的简单模型,通过它们的相互作用来达到整体行动目的。也就是说以有限状态机为中心,构成产生某单一行动的模型,我们称之为扩展的有限状态机(Augmented Finite State Machines,简称 AFSM)。通过不断地增加 AFSM,以完成所期望的动作,从而构造出智能机器人^[18]。例如,让机器人沿着墙壁自动地行走。我们分别用遗传编程和 EHW 方法实现这种 SSA。虽然二者用的都是遗传方法,但 EHW 的运行速度比遗传编程明显地快很多,并且 EHW 可以实时控制机器人。这其中,为了增加 AFSM,考虑了用 EHW 实现有限状态机的学习。因此对于这样的 SSA 问题,完全可以用基于 EHW 的硬件控制来代替原来的软件控制。

3. EHW 的自适应性

EHW 的一大特点是能适用于未知环境,即 EHW 具有“应变”能力。即使事先不知道在新的环境里硬件应该具有怎样的逻辑行为,当新的环境出现的时候 EHW 也能自动地改变其自身结构以适应新的环境。这种情况对于传统的硬件设计来说是不可能的,因为系统设计者在开始设计时,并不知道系统的逻辑行为将如何变化,设计时便无法考虑。EHW 的这一特点意味着,它代表着新一代的设计方法论。

除此之外,当硬件发生故障而不能利用环境输入信号时,EHW 能重新构造其硬件结构。在通常的硬件设计中,也能考虑一些故障诊断的硬件设计。但必须能预测故障之所在,这就使电路设计比较冗长,造成额外的负担。针对这种情况,对于可靠性较低的电路可利用 EHW 并通过自适应学习来克服上述问题。因此 EHW 具有更好的容

错性。

8.5.2 硬件进化的学习方法

下面先讨论 GAL16V8芯片的基本结构,然后介绍采用该芯片的遗传进化的学习方法。

1. GAL16V8的结构

图8.25是 GAL16V8的逻辑结构图。它有8个称为 OLMC (Output Logic Macro Cell)的逻辑宏单元,单元之间的连接方式由内部熔丝阵列决定。芯片引脚总共19个,其中1至9为输入脚,12至19为输出脚。

熔丝阵列定义了逻辑宏单元之间的连接方式,但它实际上定义的是对于逻辑宏单元输入的乘积项。我们以图8.25中的 OLMC19为例进行说明。图中熔丝阵列用连线交叉点处的黑点表示。来自输入引脚的信号,被“连接”到熔丝阵列行方向的信号线上以表示信号的加载。在熔丝阵列的第一行上有两个黑点,它表示来自第2引脚的输入信号 P_2 和来自第3引脚的信号 P_3 被加载到这一行上。当熔丝阵列内的某一行上加载了多个信号时,表示是这些信号的逻辑与(AND)操作。因此图中第一行线的输出信号为 $P_2 \cdot P_3$,并送往该线右端的 OLMC19。在 OLMC 的入口处有一些乘积项输入禁止位。如果这一位为逻辑真,则相应横线输出的乘积项原封不动地送往 OLMC。例如图中第一横线上的禁止位为真,所以信号 $P_2 \cdot P_3$ 被送往 OLMC19的输入端。如果禁止位为逻辑假,则信号被阻断。同样,图8.25中的信号 $P_4 \cdot P_5$ 也被送往 OLMC19。另一方面,逻辑宏单元 OLMC 的内部结构可如图8.26所示。它由触发器、或门、异或门和多路选择器组成。控制信号主要有3个,即 SYN、AC0和 AC1,这三位信号是结构位的一部分,因为三种信号的不同组合可以使该电路产生各种动作模式,比如寄存器模式,组合电路输出模式等。逻辑宏单元的输出可采用正逻辑或负逻辑,并由异或位 XOR 选择。所以 XOR 信号也属于结构位。当 $SYN=1, AC0=0, AC1=0, XOR=1$ 时,OLMC19可具有单元

组合电路功能,其输出为 $P_2 \cdot P_3 + P_4 \cdot P_5$, 此时 OLMC19 的内部结构如图 8.27 所示。

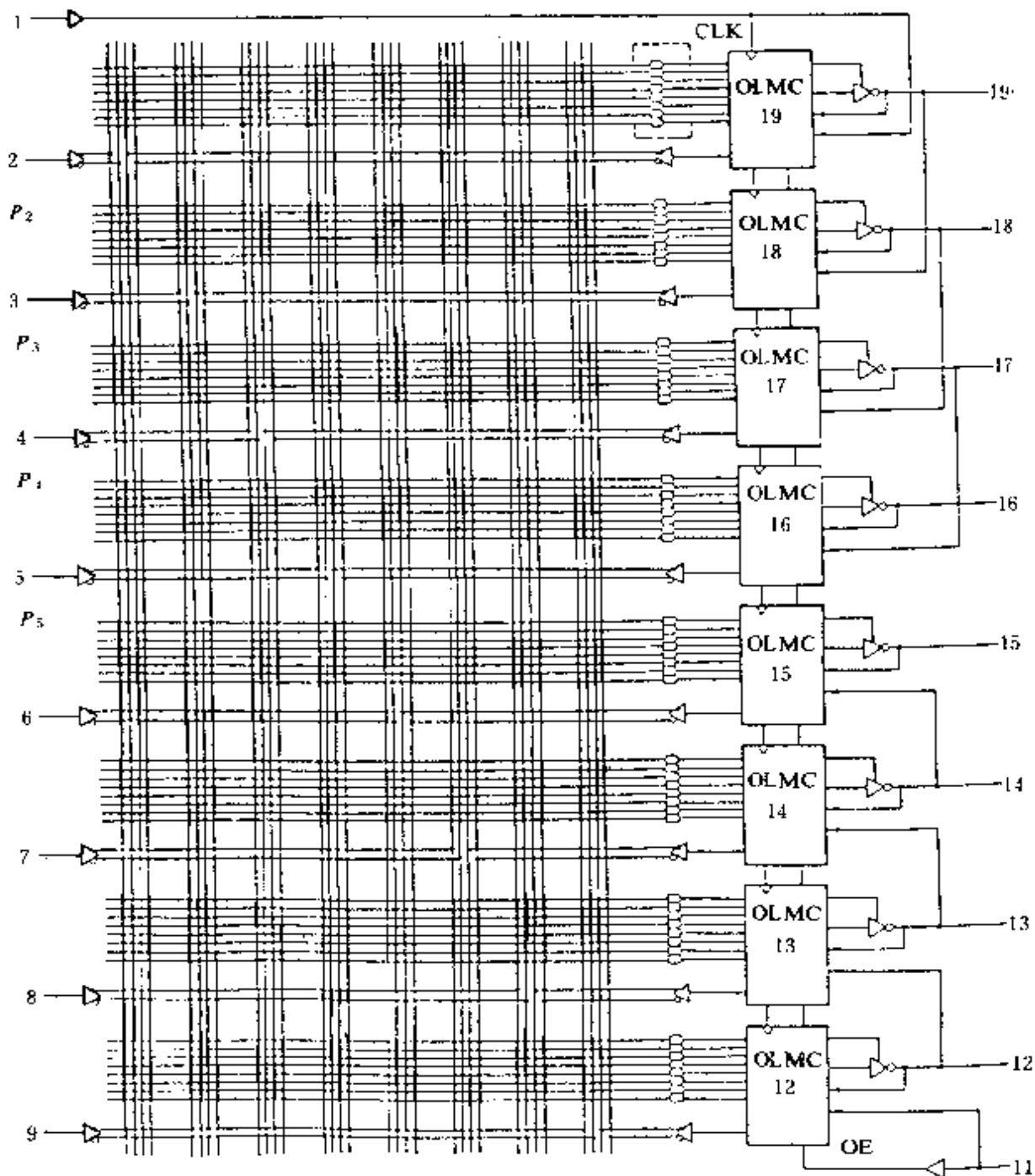


图 8.25 GAL16V8 的逻辑结构图

綜上述,在 GAL 内部,逻辑宏单元的功能和单元之间的连接方式(熔丝阵列)由结构位决定,并以此来决定 GAL 硬件的功能。

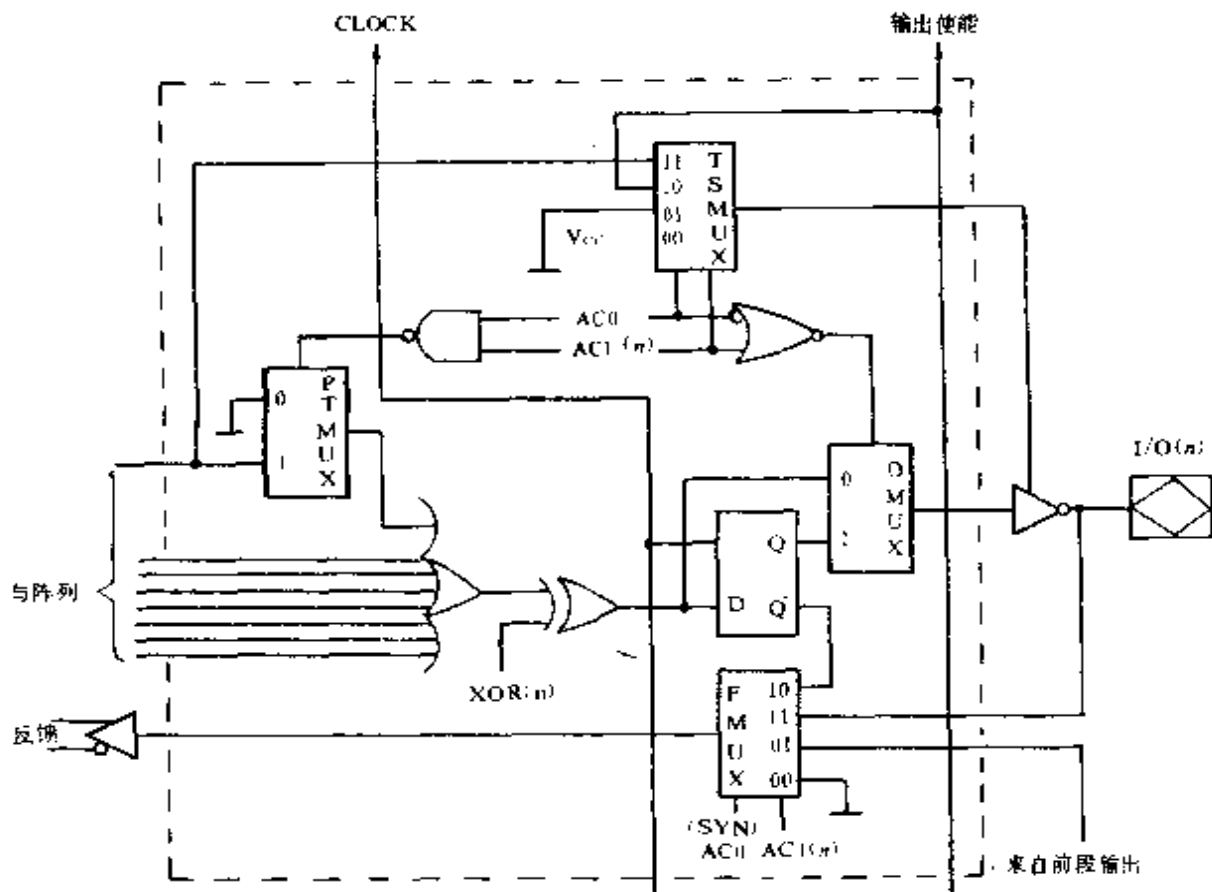


图 8.26 逻辑宏单元的内部结构

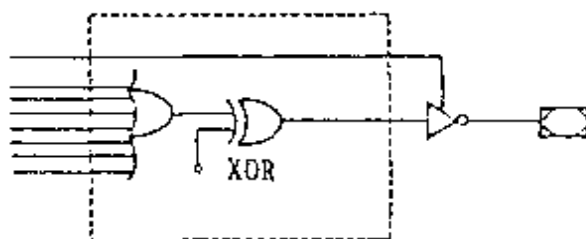


图 8.27 OLMC19的动作方式:组合电路

2. 遗传学习的方法

(1) 染色体表示。

EHW 的染色体表示与结构位(Architecture bit)相对应,也就是说,可以把每个决定逻辑功能的位序列与决定熔丝阵列模式的位序列一起看作一个染色体,我们仍以上述 OLMC19的组合功能 $Out =$

$P_2 \cdot P_3 + P_4 \cdot P_5$ 为例来加以说明。为此,我们设想由图 8.28 所示的简单 GAL 电路来实现上述功能,该电路是 4 输入 1 输出电路,图中反馈线被省略了。熔丝阵列在纵、横方向各有 8 根线,形成了 $8 \times 8 = 64$ 个网格点,如果格子点处于 ON 状态(即横线与纵线相连),则输入信号被加载到横线上。这里染色体的位数共由 76 位组成,其中包括格子点数 64 位和决定逻辑宏单元功能的 12 位(含决定逻辑乘积项的 8 位和选择逻辑宏单元功能的 4 位)。由多个这样的染色体形成群体,再用遗传算法进行操作,从而达到学习的目的。

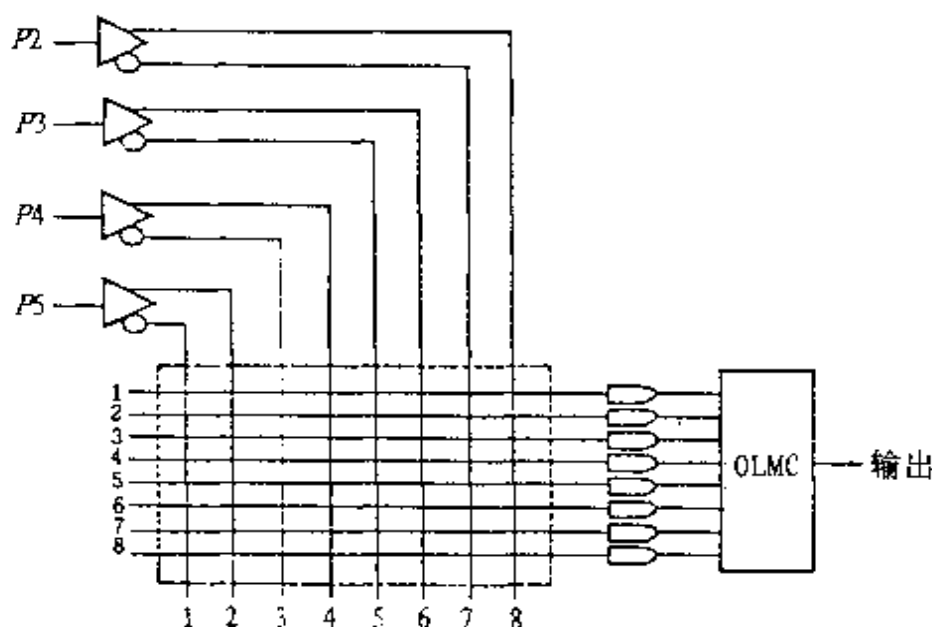


图 8.28 简单 GAL 电路

(2) 学习方法。

图 8.28 电路学习方法如图 8.29 所示。由于图中所采用的 16V8 型 GAL 芯片,可擦写的次数不多。所以,需要先用软件进行进化模拟,然后再将程序代码一次烧入 GAL 芯片,以减少 GAL 的擦写次数。这是用目前的 PLD 设计 EHW 的不足之处。但是,目前有一种可在线重写的 GAL 芯片,叫做 16Z8。用这种芯片进行 EHW 设计就方便得多。图 8.29 的学习过程是:GAL 模拟器在某一时刻接受一个染色体和来自环境的外部输入信号,并从与该染色体表示相对应的硬件

结构产生输出,然后把这种输出送往适应度评价模块,以形成一个与求解问题相对应的过程。适应度的计算是用通常的 GA 操作针对群体进行的。

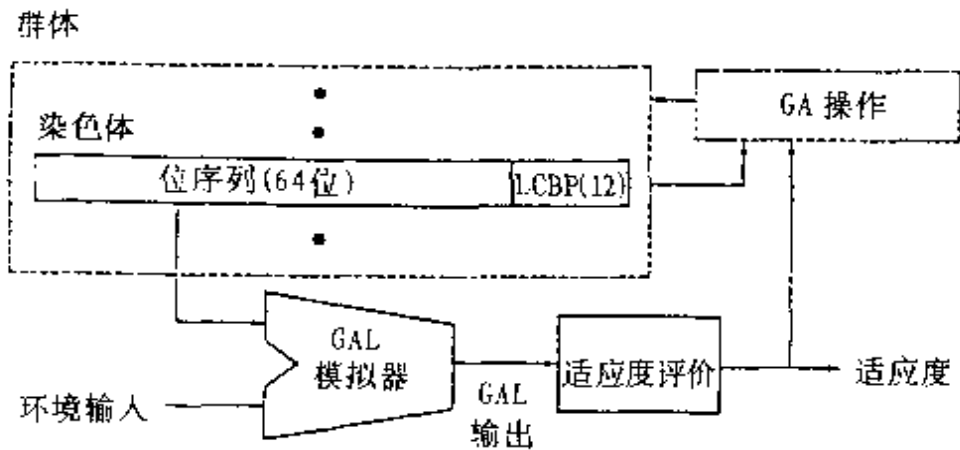


图 8.29 学习方法

8.5.3 实例

我们选择两个实例来说明硬件进化,一个是组合电路6路复用器,其输出仅由输入决定,另一个是时序电路即3位计数器,其输出由输入和状态共同决定。

1. 组合电路

图8.30是6路复用器的示意图。它有4个输入信号端口,2根地址线,一根输出线。通过指定地址位,把所选择的输入端口的数据送往输出线,通过建立 Boole 分类系统^[19],该6路复用问题可用 IF-THEN 形式的规则进行学习。把6位输入(4位数据和2位地址)作为条件部,把1位输出作为结论部。然后进行400次随机输入输出操作,也就是随机设定400条规则,并产生输入输出关系曲线。把其中能产生正确的输入输出操作的规则予以“奖励”,而对错误的操作予以惩罚。用这种方法来调整规则的“有用度”,并把这种有用度作为适应度来执行遗传算

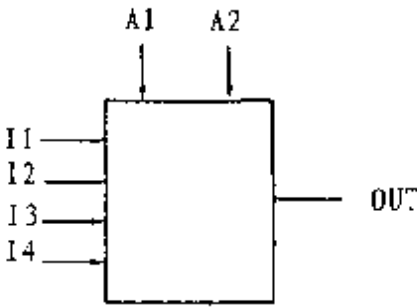


图 8.30 6路复用器

法。对于6路复用问题,只需要8条规则就能学习成功。这意味着,通过输入输出操作来学习多路复用函数所形成的规则,与传统人工智能中的规则获取是完全不一样的,因为它不需要任何先验知识。

Boole 分类系统和 EHW 都是通过遗传算法进行学习,但二者的区别在于,Boole 系统用学习结果获得规则,而 EHW 则是用学习结果获得决定 GAL 功能的逻辑单元及各单元之间的连接模式,其结果是用硬件直接表示的。

(1) 染色体表示。

多路复用问题对应单输出函数,所以用 GAL 的一个逻辑宏单元就能实现。一个 GAL 芯片往往有8个逻辑宏单元,所以这里使用一片 GAL。由于在6路复用电路中有6根输入线,参看图8.25便知, GAL 的熔丝阵列需要12根竖线。又由于一个逻辑宏单元有8根横线,故6路复用电路熔丝阵列构成的位结构有 $12 \cdot 8 = 96$ 位,再加上乘积项禁止输入位和输出极性控制位所需12位,所以一个染色体需用 $96 + 12 = 108$ 位来表示。一个染色体对应于一个 GAL 芯片结构,染色体不同,表示 GAL 芯片的结构和功能不一样。如果群体由100个染色体组成,各染色体的适应度可用下述方法确定。

对于6路复用问题总共有64个输入输出模式作为训练数据,分别给定每一个训练模型,来检查 GAL 的输出与训练数据输出一致性。如果一致,则增加该染色体的适应度,否则减少其适应度。若对于所有模式,都能找到的正确解,则该染色体的适应度为最大值64。要想获得达到100%的正确率的某个染色体,需要反复操作以下步骤:

- 1) 计算各染色体的适应度,
- 2) 用适应度的比例概率选择两个染色体,
- 3) 通过遗传算法得到两个新的染色体,
- 4) 置换原来的群体。

(2) 实验结果。

在实验中的交叉操作,大约用20%的概率进行一致交叉(uniform crossover),每位以0.1%概率进行变异,图8.31是多路复用电

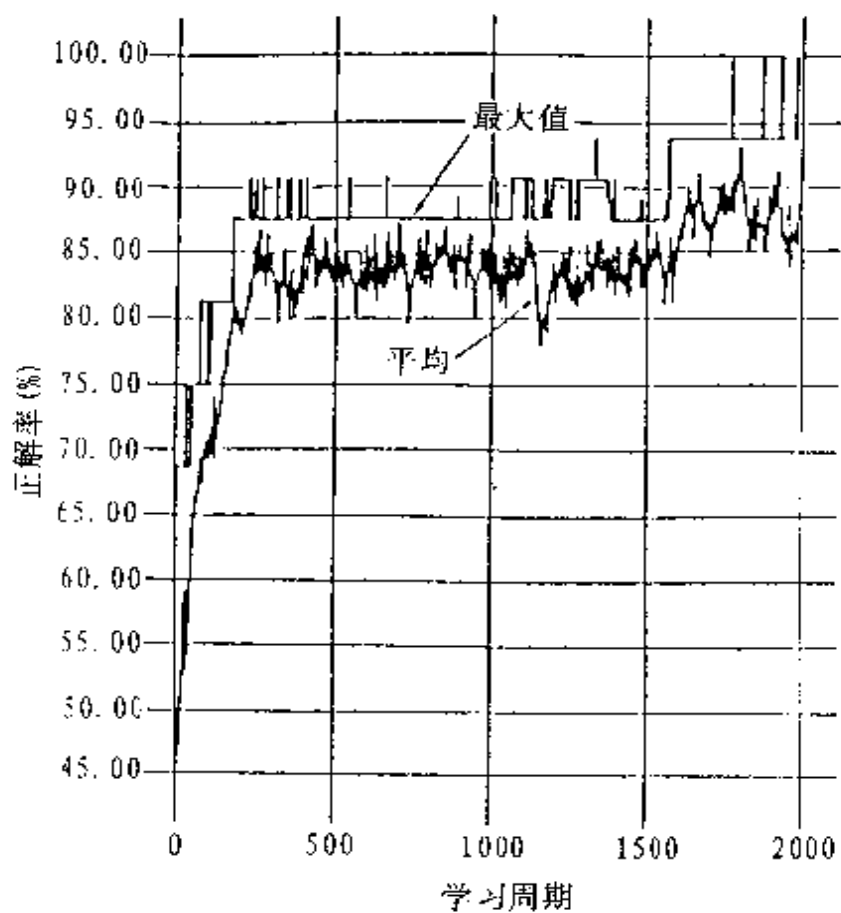


图 8.31 6路复用器的学习

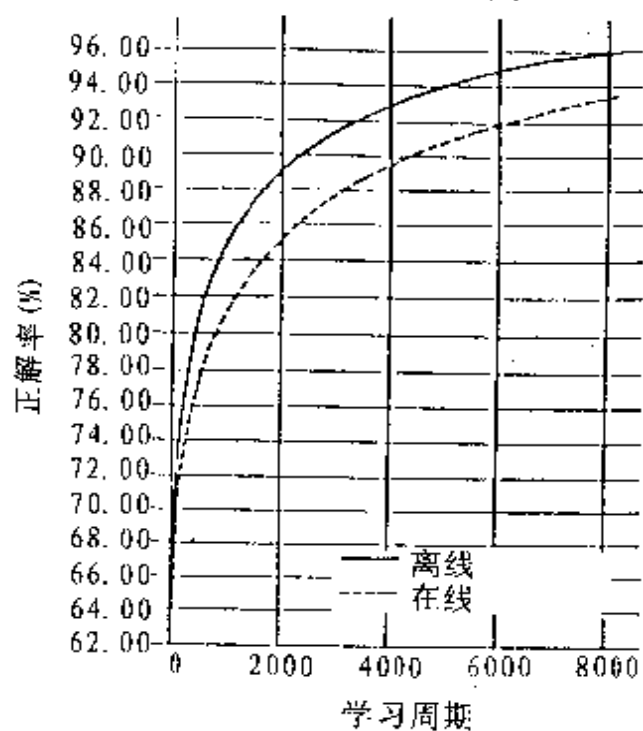


图 8.32 6路复用器时在线和离线性能

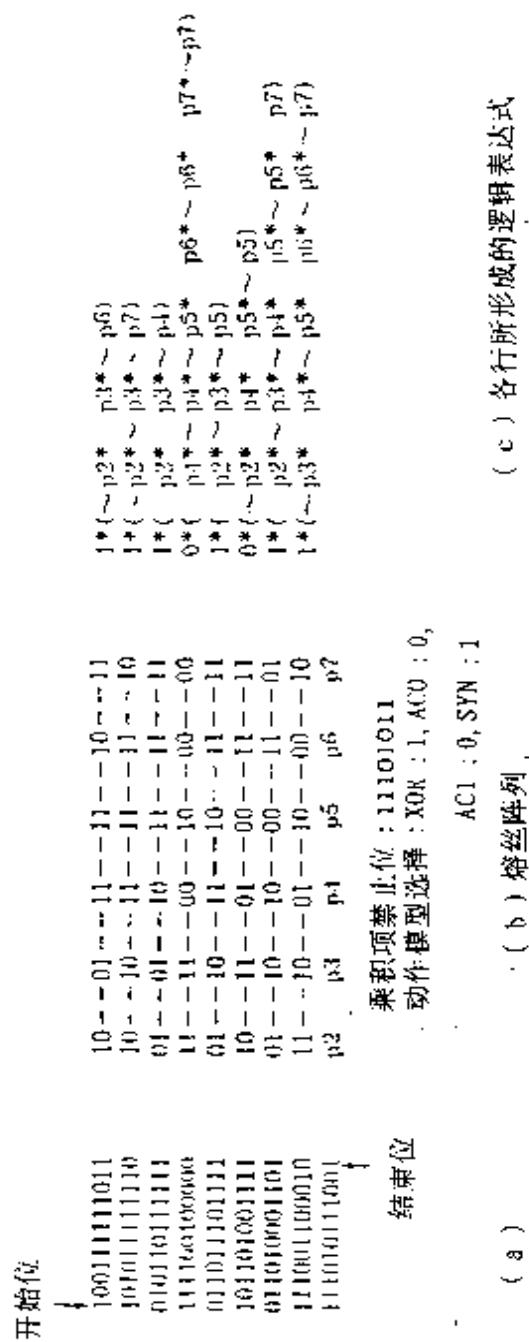


图 8.33 染色体表示

路进化的一个实验。图中表示了群体中100个个体(染色体)的平均适应度和最佳个体的适应度,大约在2000次循环以后,获得了具有100%正确率的个体。图8.32表示了使用随机数发生器产生不同的个体,并进行了10次实验操作以后的在线性能和离线性能。图8.33(a)则表示进化以后的染色体的一个例子。

如前所述,它是由乘积项禁止位,熔丝阵列位和逻辑功能确定位组成。它们的对应关系示于图8.33(b),图8.33(a)的最后一行的位序列是乘积项禁止位和逻辑功能确定位,图8.33(c)表示熔丝阵列的各行所形成的逻辑表达式,它们是各乘积项的函数,函数值将送往逻辑宏单元。表达式的第一位数字(0或者1)是乘积项输入禁止位的值,若为0则该行表达式无效。从这些行的乘积项,可以求得6路复用器的函数功能:

$$\begin{aligned} \text{out:} = & ((\bar{p}_2 \cdot p_3 \cdot \bar{p}_4) + (p_2 \cdot \bar{p}_3 \cdot \bar{p}_5) \\ & + (\bar{p}_2 \cdot p_3 \cdot \bar{p}_6) + (\bar{p}_2 \cdot \bar{p}_3 \cdot \bar{p}_7)) \end{aligned}$$

与此相对应,用 GAL 形成的6路复用电路如图8.34所示。

2. 时序电路

(1) 3位计数器。

计数器比较简单。在 EHW 中,由于含有触发器,所以3位计数器的进化也比较容易。计数器的遗传学习,由8个时钟周期的学习过程确定。GAL 的状态与作为教师信号的3位计数器的状态进行比较。这里的状态由3个有触发器的输出表示。

适应度用正解率,即所获得8个状态中有几个是正确的解来表示。群体中染色体数为100,染色体位数为157,变异概率为0.5%,交叉概率为60%时,计数器电路平均需要经过200个左右的学习周期才能完成,如图8.35所示。

(2) 有限状态机。

有限状态机,或者称为自动机,与3位计数器相比要难得多。这是因为在进化过程中,只给输入信号,看不到像计数器那样的状态。仅仅是能够根据输入知道其输出,因此有限状态机可看作一个黑盒子

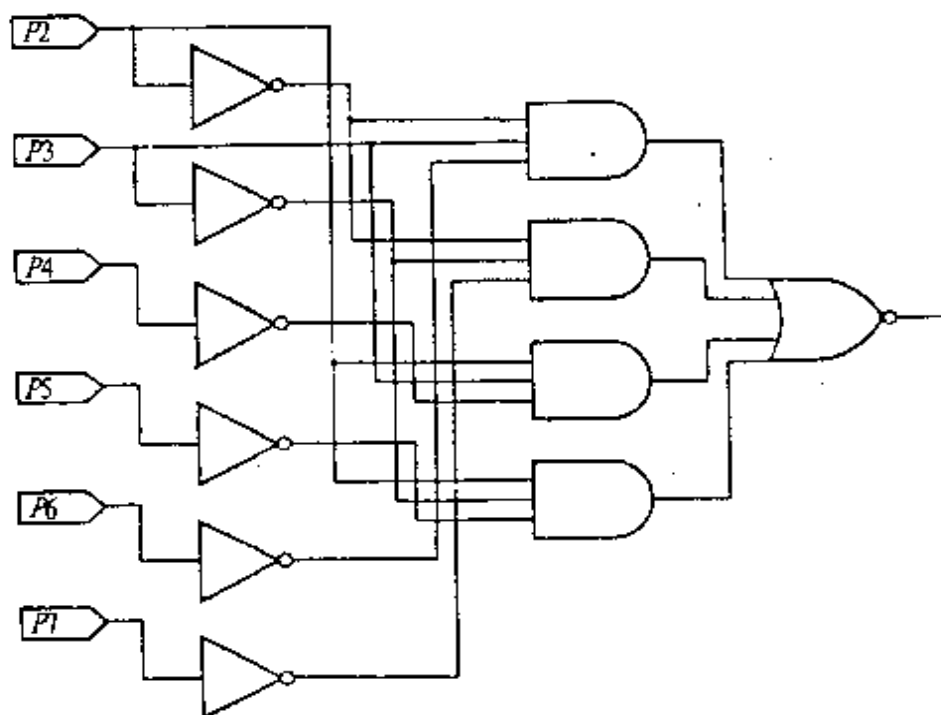


图 8.34 6路复用器的电路型

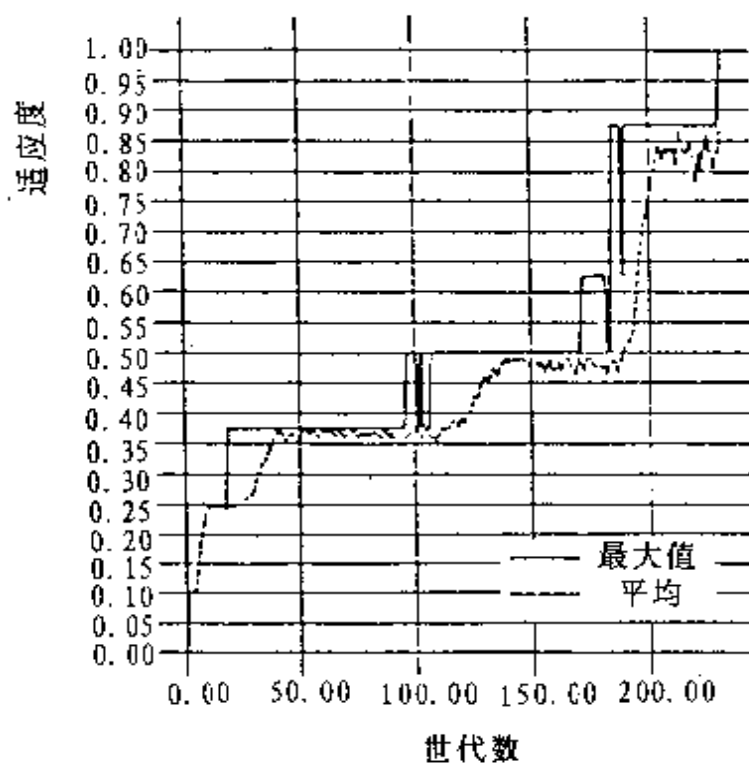


图 8.35 3位计数器的学习

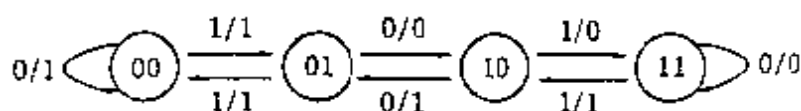


图 8.36 有限状态机的状态转换

在 GAL 上生成,此时也无法设定初始状态。有限状态机的学习按以下方式进行。设作为设计目标的有限状态机是如图8.36所示的状态转换图所表示的4状态电路。输入信号引起状态转移,同时产生输出信号。因此,GAL 芯片也应该是对相同的输入信号序列产生相同的输出信号序列。在实验过程中,随机地输入0或1,并反复1000次,对目标自动机与 GAL 芯片(实际上是 GAL 模拟器)的输出进行比较。其适应度就是正确输出的比率。通过遗传学习可达到100%。例如,当 GAL 的第 N 次输出与目标自动机不一致,适应度就是 $(N-1)/1000$ (1000是输出总数)。

图8.37是学习成功的一个例子,群体染色体数是100,染色体长度是157位,其中132位是熔丝阵列,交叉概率取20%,变异概率取

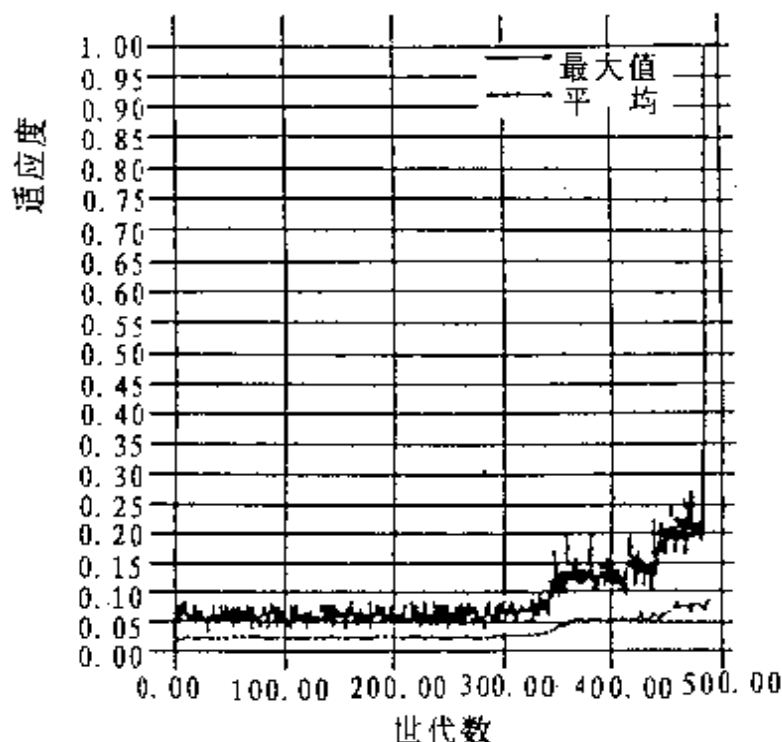


图 8.37 有限状态机的学习

0.5%，在次的试验中平均学习周期为650，最小是411个周期，最大是1303个周期。这个学习周期相对于整个搜索空间(10^{45})来说是很短的，图8.38是从学习结果得到的有限状态机的电路图。

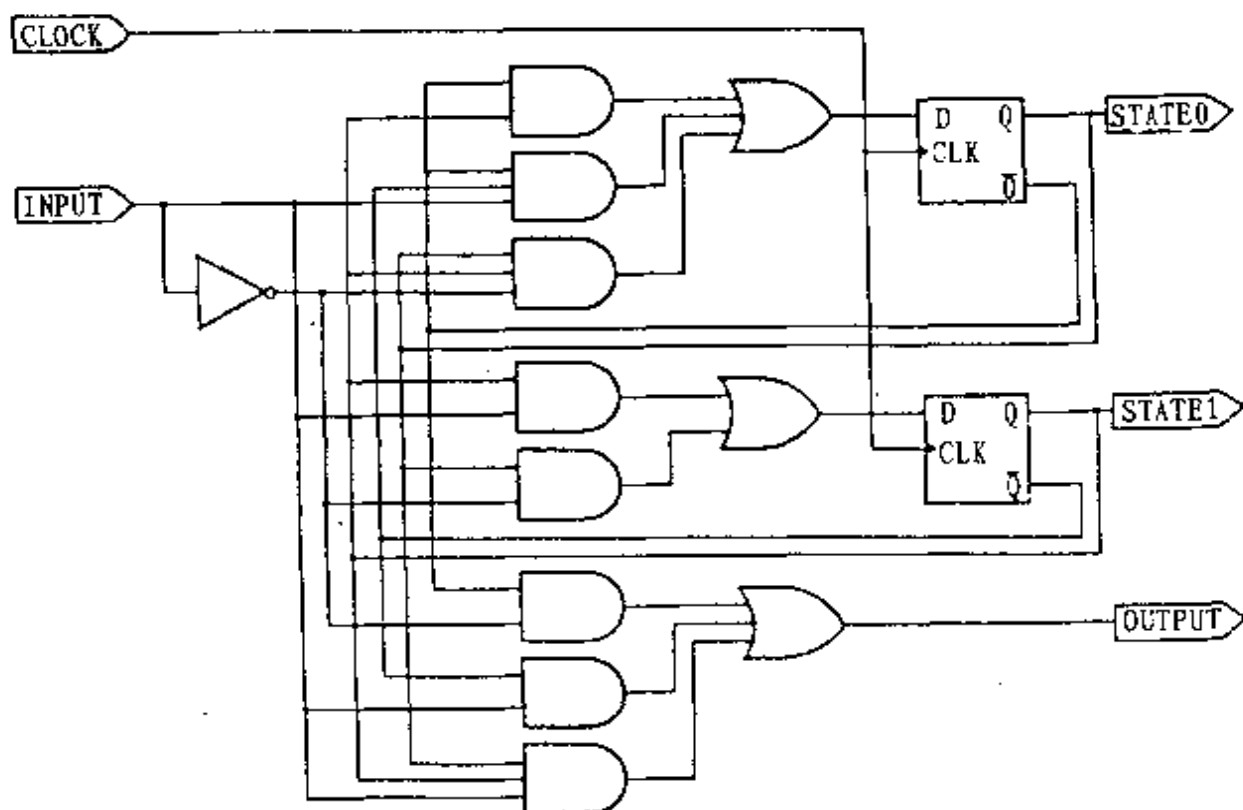


图 8.38 有限状态机(4状态)的电路图

用该电路所做实验，可得到图8.39所示的状态转换图。它表示可以并行地得到多个候选解，并且，无需很长的输入信号序列就可得到一部分候选解。

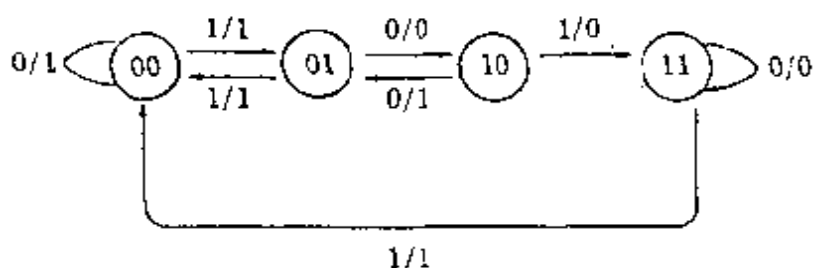


图 8.39 部分正确的有限状态机

参 考 文 献

- [1] Rosenfeld A. Digital Picture Processing. New York: Academic Press, 1976
- [2] Marr D. Vision. W H Freeman and Company, San Francisco, 1982
- [3] Geman S, Geman D. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. IEEE Trans PAMI, 1984, 6 (6): 721~741
- [4] Bhattacharjya AK, et al. Joint Solution of Low, Intermediate, and Hige-Level Vision Tasks by Evolutionary Optimization; Application to Computer Vision at Low SNR. IEEE Trans NN, 1994, 5(1): 71~94
- [5] 高橋, 北村, 堀山. Genetic Algorithms の最適制御問題への適用. 第17回「システムシンポジウム」第14回「知能システムシンポジウム」第1回「ニューテルネットワークシンポジウム」合同シンポジウム資料, 計測自動制御学会163~168, 1991
- [6] Odetayo M O, McGregor. D R Genetic Algorithm for Inducing Control Rules for a Dynamic System. Proc of CGA, 1989, 177~182
- [7] Whitley D, Ominic S, Das R. Genetic Reinforcement Learning With Multilayer Neural Networks. Proc of JCGA, 1991, 562~569
- [8] Goldberg D E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, 1986
- [9] Graham R L, Lawler E L, Lenstra J K and Rinnooy Kan A H G. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. Annals of Discrete Mathemat-

ics, 1979, 5; 287~326

- [10] Muth J F, Thompson G L. Industrial Scheduling. Prentice-Hall, Englewood Cliffs, 1963
- [11] Nakono R, Yamada T. Conventional Genetic Algorithm for Job Shop Problems Proc of 4th, Int Conf on Genetic Algorithms and Their Applications, 1991, 474~479
- [12] Syswerda G. Uniform Crossover in Genetic Algorithms. Proc of 3rd, Int Conf on Genetic Algorithms and Their Applications, 1989. 2~9
- [13] Yamada T, Nakano R. A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems. Proc of Parallel Problem Solving from Nature II, 1992. 281~290
- [14] 北野宏明. 遺伝的アルゴリズム産業図書株式会社, 1993. 263~286
- [15] Higuchi T, et al. Evolving Hardware with Genetic Learning. Proc of Simulated Adaptive Behavior, MIT Press, 1993
- [16] Spofford J J, Hintz K J. Evolving Sequential Machines in Amorphous Neural Network. in Artificial Neural Network (ed. Kohonen, et al.), Elsevier Science Publishers, 1991
- [17] Brooks. A Robot that Works; Emergent Behavior from a Carefully Evolved Network. Neural Computation, 1989
- [18] Brooks. 五味, 複数の要素行動間の競合協調により知能ロボットの行動を決めるサブサンプリング. アーキテクチャ. 日経インテリジェントシステム, 別冊1992年春号
- [19] Wilson S. Classifier System and the Animat Problem. Machine Learning, 1987, 2: 199-228

附录 A SGA 程序

本程序是一个基本的简单遗传算法示范程序,其优化目标是在区间 $[0, 2\pi]$ 上搜索函数 $\sin^2(x)$ 的最大值。该程序由选择、交叉和变异三个遗传操作及群体更新等主要算法模块以及诸如随机数发生器、输入输出、译码、适应度计算等辅助模块组成。选择机制为比例选择机制,交叉采用单点交叉策略,变异采用简单的位点变异方式。程序的输入参数为群体规模、染色体长度、交叉概率、变异概率及遗传世代数等五个遗传控制参数。其中,群体规模的取值范围为2~100之间的任意偶数,染色体长度可取不大于64的整数,交叉概率和变异概率的取值范围为 $[0.00, 1.00]$ 。输出数据包括个体序号、父序号、交叉位置、个体码串、译码值、目标函数值等栏目以及交叉次数、变异次数、世代数、群体中的最大、最小及平均适应度等统计参数。为便于观察群体进化过程,本程序采用代间中断运行方式;每按键一次,执行一代。

下面所提供的 SGA 程序清单,系采用 TURBO C 2.0 编制而成;经编译、链接后,其可执行程序可在 DOS3.3 以上的环境中运行。

SGA 程序清单:

```
#include <float.h>
#include <time.h>
#include <math.h>
#include <graphics.h>
#include <stdlib.h>

#define maxpop 100
#define maxstring 64
```

```

struct pp {unsigned char chrom[maxstring];
           float x,fitness;
           unsigned int parent1,parent2,xsite;
           };

struct pp *oldpop,*newpop,*p1;
unsigned int popsize,lchrom,gen,maxgen,nmutation,ncross,jcross,maxpp,
minpp,jrand;
float pcross,pmutation,sumfitness,avg,max,min,seed,rj[maxpop],oldrand
[maxpop];
double coef;
float objfunc(float);
void statistics();
int select();
int flip(float);
int crossover();
int mutation();
void generation();
void initialize();
void report();
void initpop();
void initdata();
void initreport();
float decode();
float random1();
void randomize1();

```

/* SGA 主程序 */

```

main()
{
long int gen,k,j;
float oldmax;
int oldmaxpp;

```

```

if (! (oldpop=(struct pp * )malloc(maxpop * sizeof(struct pp))))
    {printf("memory request failed! \n");exit(0);}
if (! (newpop=(struct pp * )malloc(maxpop * sizeof(struct pp))))
    {printf("memory request failed! \n");exit(0);}
if (! (p1=(struct pp * )malloc(sizeof(struct pp))))
    {printf("memory request failed! \n");exit(0);}
for(k=0;k<maxpop;k++) oldpop[k].chrom[0]='\0';
for(k=0;k<maxpop;k++) newpop[k].chrom[0]='\0';
gen=0;
initialize();
clrscr();
p1=newpop;
newpop=oldpop;
statistics(newpop);
report(gen);
newpop=p1;
getch();
do {
    gen=gen+1;
    randomize();
    oldmax=max;
    oldmaxpp=maxpp;
    generation();
    statistics(newpop);
    if(max<oldmax)
        {for(j=0;j<lchrom;j++)
            newpop[minpp].chrom[j]=oldpop[oldmaxpp].chrom[j];
            newpop[minpp].x=oldpop[oldmaxpp].x;
            newpop[minpp].fitness=oldpop[oldmaxpp].fitness;
            statistics(newpop);
        }
}

```

```

    report(gen);
    p1=oldpop;
    oldpop=newpop;
    newpop=p1;
    getch();
}while (gen<maxgen);
free(p1);
free(oldpop);
free(newpop);
exit(0);
}

/* 个体适应度计算 */
float objfunc(float x1)
{float y;
  y=3.1415926*x1;
  y=sin(2.0*y);
  return y*y;
}

/* 群体适应度统计 */
void statistics(pop)
struct pp *pop;
{int j;
  sumfitness=pop[0].fitness;
  min=pop[0].fitness;
  max=pop[0].fitness;
  maxpp=0;
  minpp=0;
  for (j=1;j<popsiz;j++)
    {sumfitness=sumfitness+pop[j].fitness;
     if (pop[j].fitness>max)

```

```

        {max=pop[j]. fitness;
         maxpp=j;
        }
    if (pop[j]. fitness<min)
        {min=pop[j]. fitness;
         minpp=j;
        }
    }
    avg=sumfitness/(float)popsiz;
}

/* 群体更新 */
void generation()
{unsigned int j,mate1,mate2;
j=0;
do{
    mate1=select();
    mate2=select();
    crossover(oldpop[mate1]. chrom,oldpop[mate2]. chrom,j);
    newpop[j]. x=(float)decode(newpop[j]. chrom);
    newpop[j]. fitness=objfunc(newpop[j]. x);
    newpop[j]. parent1=mate1;
    newpop[j]. parent2=mate2;
    newpop[j]. xsite=jcross;
    newpop[j+1]. x=(float)decode(newpop[j+1]. chrom);
    newpop[j+1]. fitness=objfunc(newpop[j+1]. x);
    newpop[j+1]. parent1=mate1;
    newpop[j+1]. parent2=mate2;
    newpop[j+1]. xsite=jcross;
    j=j+2;
}while (j<popsiz);
}

```

```

/* 控制参数输入 */
void initdata()
{unsigned int ch,j;
clrscr();
printf("-----\n");
printf("A Simple Genetic Algorithm - SGA\n");
printf(" (c) X. F. CHEN JUNE 1994 \n");
printf(" All Rights Reserved \n");
printf("-----\n");
pause();clrscr();
printf(" ***** SGA DATA ENTRY AND INITIALIZATION ***** \n");
printf("\n");
printf("Enter population size -----> "); scanf("%d",&popsize);
printf("Enter chromosome length -----> "); scanf("%d",&lchrom);
printf("Enter max. generations -----> "); scanf("%d",&maxgen);
printf("Enter crossover probability-----> "); scanf("%f",&pcross);
printf("Enter mutation probability-----> "); scanf("%f",&pmutation);
clrscr();
randomize();
randomize1();
nmutation=0;
ncross=0;
}

/* 初始化信息输出 */
void initreport()
{ int j,k;
printf("-----\n");
printf(" A Simple Genetic Algorithm - SGA\n");
printf(" (c) X. F. CHEN JUNE 1994 \n");

```

```

printf("    All Rights Reserved \n");
printf("    -----\n");
printf("SGA Parameters\n");
printf("    -----\n");
printf("\n");
printf(" Population size (popsize)    = %d\n",popsize);
printf(" Chromosome length (lchrom)    = %d\n",lchrom);
printf(" Maximum # of generation(maxgen) = %d\n",maxgen);
printf(" Crossover probability (pcross) = %f\n",pcross);
printf(" Mutation probability (pmutation) = %f\n",pmutation);
printf(" -----\n");
printf("\n");
printf(" Initial Population Maximum Fitness = %f\n",max);
printf(" Initial Population Average Fitness = %f\n",avg);
printf(" Initial Population Minimum Fitness = %f\n",min);
printf(" Initial Population Sum of Fitness = %f\n",sumfitness);
pause();
}

```

/* 生成初始群体 */

```

void initpop()
{int j,j1;
randomize();
for(j=0;j<popsize;j++)
    {for(j1=0;j1<lchrom;j1++)
        oldpop[j].chrom[j1]=random(2);
    oldpop[j].x=(float)decode(oldpop[j].chrom);
    oldpop[j].fitness=objfunc(oldpop[j].x);
    oldpop[j].parent1=0;
    oldpop[j].parent2=0;
    oldpop[j].xsite=0;
    }
}

```

```

}

/* 初始化 */
void initialize()
{
    initdata();
    coef=pow(2.00,lchrom)-1.0;
    initpop();
    statistics(oldpop);
    initreport();
}

/* 数据输出 */
void report(int gen)
{int k,j;
  for (j=0;j<79;j++) printf(" * ");
  printf("\n");
  printf("                Population Report\n");
  printf("                Generation %3d\n",gen);
  printf(" # parents xsite      string          x  fitness");
  printf("\n");
  for(j=0;j<79;j++) printf(" + ");
  printf("\n");
  for (j=0 ; j<popsize;j++)
      {printf(" %2d ) ( %2d ,",j,newpop[j].parent1);
        printf(" %2d ) %2d",newpop[j].parent2,newpop[j].xsite);
        for(k=0;k<lchrom;k++)
            printf(" %d",newpop[j].chrom[k]);
        printf(" %12.1f ",newpop[j].x);
        printf(" %6.4f -New. \n",newpop[j].fitness);
      }
  for(k=0;k<79;k++) printf(" * ");

```



```

printf("\n");
printf("RESULT: GEN: %3d ",gen);
printf("AVG= %8.4f MIN= %8.4f MAX= %8.4f \n",avg,min,max);
printf("ncross= %4d nmutation= %4d\n",ncross,nmutation);
}

```

/* 译码 */

```

float decode(unsigned char *pp)
{int j;
 float tt,ttl;
 ttl=1.0;
 tt=0.0;
 for (j=lchrom-1;j>=1;j--)
 {if (pp[j]) tt=tt+ttl;
  ttl=2.0*ttl;
 }
 tt=tt/coef;
 return tt;
}

```

/* 延时 */

```

pause()
{int j,j1;
 int x1;
 x1=0;
 for (j=1;j<=25;j++)
 {for (j1=1;j1<2;j1++) x1=x1+1;
 }
 return x1;
}

```

/* 选择操作 */

```

int select()
{double rand1,partsum;
  int j;
  partsum=0.0;
  j=0;
  rand1=random1()*sumfitness;
  do {
    partsum=partsum+oldpop[j].fitness;
    j=j+1;
  }while ((partsum < rand1) && (j < popsize));
  return j-1;
}

```

/* 变异操作 */

```

int mutation(ch)
unsigned char ch;
{int mutate,j;
  mutate=flip(pmutation);
  if (mutate)
    {nmutation=nmutation+1;
      if (ch==0;
        else ch=1;
      }
  if(ch) return 1;
    else return 0;
}

```

/* 交叉操作 */

```

int crossover(unsigned char *parent1,unsigned char *parent2,int k5)
{int i,j,j1;
  if (flip(pcross))
    {jcross=random(ichrom-1);

```

```

        ncross = ncross - 1;
    }
    else jcross = lchrom;
    if (jcross != lchrom)
    {
        for (j = 0; j < jcross; j++)
        {
            newpop[k5].chrom[j] = mutation(parent1[j]);
            newpop[k5 + 1].chrom[j] = mutation(parent2[j]);
        }
        for (j = jcross; j < lchrom; j++)
        {
            newpop[k5].chrom[j] = mutation(parent2[j]);
            newpop[k5 + 1].chrom[j] = mutation(parent1[j]);
        }
    }
    else
    {
        for (j = 0; j < lchrom; j++)
        {
            newpop[k5].chrom[j] = mutation(parent1[j]);
            newpop[k5 + 1].chrom[j] = mutation(parent2[j]);
        }
    }
    return 1;
}

```

/* 重启动随机数发生器 */

```

void randomize1()
{
    int i;
    randomize();
    for (i = 0; i < lchrom; i++)
        oldrand[i] = random(30001) / 30000.0;
    jrand = 0;
}

```

/* 随机数发生器 */

```

float random1()
{
    jrand=jrand+1;
    if (jrand>=lchrom)
        {jrand=0;
         randomize1();
        }
    return olcrand[jrand];
}

/* 贝努利试验 */
int flip(float probability)
{
    float ppp;
    ppp=random(20001)/20000.0;
    if(ppp<=probability) return 1;
    return 0;
}

```

附录 B TSP 程序

本程序是一个依据遗传算法求解货郎担问题(TSP)的应用程序。该程序由选择、交叉、逆转和变异四个遗传操作及群体更新等主要算法模块以及诸如随机数发生器、控制参数输入、群体初始化、图文数据输出、译码、适应度计算等辅助模块组成。在本程序中,TSP问题的城市坐标是由随机数发生器产生的,遗传算法的选择机制为比例选择机制,交叉采用部分匹配交叉策略,变异采用随机多次对换方式,逆转操作采用单向连续多次逆转方法,群体更新采用有覆盖的代间更新方式。程序的输入参数为群体规模、染色体长度、交叉概率、变异概率及遗传世代数等五个遗传控制参数以及存储该程序运算结果的文件名。其中,群体规模的取值范围为2~100之间的任意偶数,染色体长度(TSP的城市数)可取不大于100的整数,交叉概率和变异概率的取值范围为 $[0.00, 1.00]$ 。显示器输出的图文数据包括交叉次数、变异次数、世代数、群体中的最大、最小及平均适应度、当前最佳路径的长度、程序运行时间等统计参数及当前最佳路径图;文件存储的数据包括 TSP 问题的城市位置(平面坐标)、历代最佳路径长度以及优化的最终结果等。

下面所提供的求解 TSP 问题的 GA 程序清单,系采用 TURBO C 2.0 编制而成;经编译、链接后,其可执行程序可在 DOS3.3 以上的环境中运行。

TSP 程序清单:

```
#include <stdio.h>
#include <stdlib.h>
```

```

#include <math.h>
#include <alloc.h>
#include <conio.h>
#include <float.h>
#include <time.h>
#include <graphics.h>
#include <bios.h>

#define maxpop 100
#define maxstring 100

struct pp {unsigned char chrom[maxstring];
           float x,fitness;
           unsigned int parent1,parent2,xsite;
           };
struct pp * oldpop, * newpop, * p1;
unsigned int popsize,lchrom,gen,maxgen,col_min,jrand;
unsigned int nmutation,ncross,jcross,maxpp,minpp,maxxy;
float pcross, pmutation, sumfitness, avg, max, min, seed, maxold, oldrand
[maxstring];
unsigned char x[maxstring],y[maxstring];
float * dd,ff,maxdd,refpd,fm[201];
FILE * fp, * fp1;
float objfunc(float);
void statistics();
    int select();
    int flip(float);
    int crossover();
void generation();
void initialize();
void report();
float decode();

```

```

void crtinit();

/* 主程序 */
main()
{unsigned int gen,k,j,tt;
 char fname[10];
 float ttt;
 clrscr();
 col_min=0;
 if (! (oldpop=(struct pp *)farmalloc(maxpop * sizeof(struct pp))))
     {printf("memory request failed! \n");exit(0);}
 if (! (dd=(float *)farmalloc(maxstring * maxstring * sizeof(float))))
     {printf("memory request failed! \n");exit(0);}
 if (! (newpop=(struct pp *)farmalloc(maxpop * sizeof(struct pp))))
     {printf("memory request failed! \n");exit(0);}
 if (! (p1=(struct pp *)farmalloc(sizeof(struct pp))))
     {printf("memory request failed! \n");exit(0);}
 for(k=0;k<maxpop;k++) oldpop[k].chrom[0]='\0';
 for(k=0;k<maxpop;k++) newpop[k].chrom[0]='\0';
 printf("Enter Result Data Filename: ");
 gets(fname);
 if (! (fp=fopen(fname,"w+")))
     {printf("cannot open file\n");exit(1);}

 gen=0;
 randomize();
 initialize();

 fputs("This is result of the tsp problem:",fp);
 fprintf(fp,"cities: %2d psize: %3d Ref. Tsp path: %f\n",lchrom,popsize,
 refpd);
 fprintf(fp,"Pc: %f Pm: %f Seed: %f\n",pcross,pmutation,seed);

```

```

fprintf(fp,"X site:\n");
for(k=0;k<lchrom;k++)
    {if ((k%16) == 0) fprintf(fp,"\n");
      fprintf(fp," %5d",x[k]);
    }
fprintf(fp,"\n Y site:\n");
for(k=0;k<lchrom;k++)
    {
        if ((k%16) == 0) fprintf(fp,"\n");
        fprintf(fp," %5d",y[k]);
    }
fprintf(fp,"\n");

crtinit();
statistics(oldpop);
report(gen,oldpop);
getch();
maxold = min;
fm[0]=100.0 * oldpop[maxpp].x/ff;
do {
    gen=gen+1;
    generation();
    statistics(oldpop);
    if(max>maxold)
        {maxold=max;
          co_min=0;
        }
    fm[gen%200]=100.0 * oldpop[maxpp].x/ff;
    report(gen,oldpop);
    gotoxy(30,25);
    ttt=clock()/18.2;
    tt=ttt/60;
} while (tt<400)

```



```

printf("Run Clock: %2d: %2d: %4.2f", tt/60, tt%60, ttt-tt*60.0);
printf("Min = %6.4f Nm: %d\n", min, co_min);
}while ((gen<100)&&(!bioskey(1)));
printf("\n gen = %d", gen);
do {
    gen=gen+1;
    generation();
    statistics(oldpop);
    if(max > maxold)
        {maxold = max;
         co_min = 0;
        }

    fm[gen%200] = 100.0 * oldpop[maxpp].x/ff;
    report(gen, oldpop);
    if((gen%100) == 0) report(gen, oldpop);
    gotoxy(30, 25);
    ttt = clock()/18.2;
    tt = ttt/60;
    printf("Run Clock: %2d: %2d: %4.2f", tt/60, tt%60, ttt-tt*60.0);
    printf("Min = %6.4f Nm: %d\n", min, co_min);
}while ((gen<maxgen)&&(!bioskey(1)));

getch();
for (k=0; k<lchrom; k++)
{
    if ((k%16) == 0) fprintf(fp, "\n");
    fprintf(fp, "%5d", oldpop[maxpp].chrom[k]);
}
fprintf(fp, "\n");

fclose(fp);
free(dd);

```

```

farfree(p1);
farfree(oldpop);
farfree(newpop);
restorecrtmode();
exit(0);
}

```

/* 适应度计算 */

```

float objfunc(float x1)
{float y;
 y:=100.0 * ff/x1;
 return y;
}

```

/* 适应度统计 */

```

void statistics(pop)
struct pp * pop;
{int j;
 sumfitness = pop[0].fitness;
 min        = pop[0].fitness;
 max        := pop[0].fitness;
 maxpp:=0;
 minpp:=0;
 for (j=1 ;j<popsize;j++)
 {sumfitness = sumfitness + pop[j].fitness;
  if (pop[j].fitness>max)
   {max = pop[j].fitness;
    maxpp:=j;
   }
  if (pop[j].fitness<min)
   {min = pop[j].fitness;
    minpp:=j;
   }
 }
}

```

• 402 •

```

    }
}

avg=sumfitness/(float)popsiz;
}

/* 群体更新 */
void generation()
{
unsigned int k,j,j1,j2,i1,i2,mate1,mate2;
float f1,f2;
j=0;
do{
    mate1=select();
    pp: mate2=select();
    if (mate1==mate2) goto pp;
    crossover(oldpop[mate1].chrom,oldpop[mate2].chrom,j);
    newpop[j].x=(float)decode(newpop[j].chrom);
    newpop[j].fitness=objfunc(newpop[j].x);
    newpop[j].parent1=mate1;
    newpop[j].parent2=mate2;
    newpop[j].xsite=jcross;
    newpop[j+1].x=(float)decode(newpop[j+1].chrom);
    newpop[j+1].fitness=objfunc(newpop[j+1].x);
    newpop[j+1].parent1=mate1;
    newpop[j+1].parent2=mate2;
    newpop[j+1].xsite=jcross;
    if(newpop[j].fitness>min)
        {for(k=0;k<lchrom;k++)
            oldpop[minpp].chrom[k]=newpop[j].chrom[k];
            oldpop[minpp].x=newpop[j].x;
            oldpop[minpp].fitness=newpop[j].fitness;
            co_min++;
        }
    }
}

```

```

        return;
    }
    if(newpop[j+1].fitness>min)
        {for (k=0;k<lchrom;k++)
            oldpop[minpp].chrom[k]=newpop[j+1].chrom[k];
        oldpop[minpp].x=newpop[j+1].x;
        oldpop[minpp].fitness=newpop[j+1].fitness;
        co.min++;
        return;
    }
    j=j+2;
}while (j<popsize);
}

/* 控制参数输入 */
void initdata()
{unsigned int ch,j;
 clrscr();
 printf("-----\n");
 printf("A Simple Genetic Algorithm - SGA\n");
 printf(" (c) X. F. CHEN JUNE 1994 \n");
 printf(" All Rights Reserved \n");
 printf("-----\n");
 pause();clrscr();
 printf(" ***** SGA DATA ENTRY AND INITIALIZATION ***** \n");
 printf("\n");
 printf("Enter population size -----> "); scanf("%d",&popsize);
 printf("Enter chromosome length -----> "); scanf("%d",&lchrom);
 printf("Enter max. generations -----> "); scanf("%d",&maxgen);
 printf("Enter crossover probability---> "); scanf("%f",&pcross);
 printf("Enter mutation probability---> "); scanf("%f",&pmutation);

```

```

randomize1();
clrscr();
nmutation=0;
ncross=0;
;

/* 初始数据输出 */
void initreport()
{ int j,k;
  printf(" -----\\n");
  printf(" A Simple Genetic Algorithm - SGA\\n");
  printf(" (c) X. F. CHEN JUNE 1994 \\n");
  printf(" All Rights Reserved \\n");
  printf(" -----\\n");
  printf(" SGA Parameters\\n");
  printf(" -----\\n");
  printf("\\n");
  printf(" Population size (popsize) = %d\\n ",popsize);
  printf(" Chromosome length (lchrom) = %d\\n ",lchrom);
  printf(" Maximum # of generation(maxgen) = %d\\n ",maxgen);
  printf(" Crossover probability (pcross) = %f\\n ",pcross);
  printf(" Mutation probability (pmutation) = %f\\n ",pmutation);
  printf(" Initial Generation Statistics \\n");
  printf(" -----\\n");
  printf("\\n");
  printf(" Initial Population Maximum Fitness = %f\\n ",max);
  printf(" Initial Population Average Fitness = %f\\n ",avg);
  printf(" Initial Population Minimum Fitness = %f\\n ",min);
  printf(" Initial Population Sum of Fitness = %f\\n ",sumfitness);
}

/* 群体初始化 */

```

```

void initpop()
{
    unsigned char j1;
    unsigned int k5,i1,i2,j,i,k,j2,j3,j4,p5[maxstring];
    float f1,f2;
    j=0;
    for (k=0;k<lchrom;k++)
        oldpop[j].chrom[k]=k;
    for (k=0;k<lchrom;k++)
        p5[k]=oldpop[j].chrom[k];
    randomize();
    for (;j<popsize;j++)
    {
        j2=random(lchrom);
        for(k=0;k<j2+20;k++)
        {
            j3=random(lchrom);
            j4=random(lchrom);
            j1=p5[j3];
            p5[j3]=p5[j4];
            p5[j4]=j1;
        }
        for(k=0;k<lchrom;k++)
            oldpop[j].chrom[k]=p5[k];
    }

    for(k=0;k<lchrom;k++)
        for(j=0;j<lchrom;j++)
            dd[k*lchrom+j]=hypot(x[k]-x[j],y[k]-y[j]);
    for(j=0;j<popsize;j++)
    {
        oldpop[j].x=(float)decode(oldpop[j].chrom);
        oldpop[j].fitness=objfunc(oldpop[j].x);
        oldpop[j].parent1=0;
        oldpop[j].parent2=0;
        oldpop[j].xsite=0;
    }
}

```

```

    }
}

/* 初始化 */
void initialize()
{int k,j,minx,miny,maxx,maxy;
  initdata();
  minx=0;miny=0;
  maxx=0;maxy=0;
  for(k=0;k<lchrom;k++)
    {x[k]=rand();
     if(x[k]>maxx) maxx=x[k];
     if(x[k]<minx) minx=x[k];
     y[k]=rand();
     if(y[k]>maxy) maxy=y[k];
     if(y[k]<miny) miny=y[k];
    }
  if((maxx-minx)>(maxy-miny)) {maxxy=maxx-minx;}
                           else {maxxy=maxy-miny;}
  maxdd=0.0;
  for(k=0;k<lchrom;k++)
    for(j=0;j<lchrom;j++)
      {dd[k*lchrom+j]=hypot(x[k]-x[j],y[k]-y[j]);
       if(maxdd<dd[k*lchrom+j]) maxdd=dd[k*lchrom+j];
      }
  refpd=dd[lchrom-1];
  for(k=0;k<lchrom-1;k++)
    refpd=refpd+dd[k*lchrom+k+1];
  for(j=0;j<lchrom;j++)
    dd[j*lchrom+j]=4.0*maxdd;
  ff=(0.765*maxxy*pow(lchrom,0.5));
  minpp=0;

```

```

min=dd[lchrom-1];
for(j=0;j<lchrom-1;j++)
    {if(dd[lchrom*j+lchrom-1]<min)
        {min=dd[lchrom*j+lchrom-1];
            minpp=j;
        }
    }
initpop();
statistics(oldpop);
initreport();

/* 图文输出及数据存储 */
void report(int gen,struct pp *pop)
{int k,ix,iy,jx,jy;
    unsigned int tt;
    float ttt;
    cleardevice();
    gotoxy(1,1);
    printf("Cities: %4d Para_size: %4d Maxgen: %4d Ref_tour: %f\n",
            lchrom,popsize,maxgen,refpd);
    printf("Ncross: %4d Nmutation: %4d Rungen: %4d AVG = %8.4f MIN = %8.4f\n",
            ncross,nmutation,l,avg,min);
    printf("Ref. co_minpath: %6.4f Minpath length: %10.4f Ref-co-tour: %f\n",
            pop[maxpp].x/maxxy,pop[maxpp].x,
            ff);
    printf("Co_minpath : %6.4f Maxfitness: %10.8f ",
            100.0 * pop[maxpp].x/ff, pop
            [maxpp].fitness);
    ttt=clock()/18.2;

    • 408 •

```



```

tt=ttt/60;
printf("Run Clock: %2d:%2d:%4.2f\n",tt/60,tt%60,ttt-tt*60.0);
setcolor(1%15+1);
for(k=0;k<lchrom-1;k++)
    {ix=x[pop[maxpp].chrom[k]];
    iy=y[pop[maxpp].chrom[k]]+110;
    jx=x[pop[maxpp].chrom[k+1]];
    jy=y[pop[maxpp].chrom[k+1]]+110;
    line(ix,iy,jx,jy);
    putpixel(ix,iy,RED);
    }
ix=x[pop[maxpp].chrom[0]];
iy=y[pop[maxpp].chrom[0]]+110;
jx=x[pop[maxpp].chrom[lchrom-1]];
jy=y[pop[maxpp].chrom[lchrom-1]]+110;
line(ix,iy,jx,jy);
putpixel(jx,jy,RED);
setcolor(11);
outtextxy(ix,iy,"*");
setcolor(12);
for(k=0;k<1%200;k++)
    {ix=k-280;
    iy=366-fm[k]/3;
    jx=ix+1;
    jy=366-fm[k+1]/3;
    line(ix,iy,jx,jy);
    putpixel(ix,iy,RED);
    }
fprintf(fp,"GEN:%3d ",l);
fprintf(fp,"Minpath: %f Maxfitness: %f ",pop[maxpp].x,pop[maxpp].fitness);
fprintf(fp," Clock: %2d:%2d:%4.2f\n",tt/60,tt%60,ttt-tt*60.0);

```

```
}
```

```
/* 译码 */
```

```
float decode(unsigned char * pp)
{int j,k,l;
 float tt;
 tt=dd[pp[0]*lchrom+pp[lchrom-1]];
 for(j=0;j<lchrom-1;j++)
 { tt=tt+dd[pp[j]*lchrom+pp[j+1]]; }
 l=0;
 for(k=0;k<lchrom-1;k++)
 for(j=k+1;j<lchrom;j++)
 {if(pp[j]==pp[k]) l++;}
 return tt+4*l*maxdd;
}
```

```
/* 设置屏幕显示方式 */
```

```
void crtinit()
{int driver,mode;
 struct palettetype p;
 driver=DETECT;
 mode=0;
 initgraph(&driver,&mode,"");
 cleardevice();
}
```

```
/* 选择 */
```

```
int select()
{double rand1,partsum;
 float r1;
 int j;
 partsum=0.0;
```

```
• 410 •
```

```

j=0;
rand1=random1()*sumfitness;
do {
    partsum=partsum+oldpop[j].fitness;
    j=j+1;
}while ((partsum < rand1) && (j < popsize));
return j-1;
}

```

/* 交叉与变异 */

```

int crossover(unsigned char *parent1,unsigned char *parent2,int k5)
{int k,j,mutate,i1,i2,j5;
 int j1,j2,j3,s0,s1,s2;
 unsigned char jj,ts1[maxstring],ts2[maxstring];
 float f1,f2;
 s0=0;s1=0;s2=0;
 if (flip(pcross))
 {jcross=random(lchrom-1);
  j5=random(lchrom-1);
  ncross=ncross+1;
  if (jcross>j5) {k=jcross;jcross=j5;j5=k;}
 }
 else jcross=lchrom;
 if (jcross1=lchrom)
 {s0=1;
  k=0;
  for (j=jcross;j<j5;j++)
  {ts1[k]=parent1[j];
   ts2[k]=parent2[j];
   k++;
  }
  j3=k;

```

```

for(j=0;j<lchrom;j++)
    {j2=0;
    while((parent2[j]!=ts1[j2])&&(j2<k)){j2++;}
    if (j2==k)
        {ts1[j3]=parent2[j];
        j3++;
        }
    }
j3=k;
for(j=0;j<lchrom;j++)
    {j2=0;
    while((parent1[j]!=ts2[j2])&&(j2<k)){j2++;}
    if (j2==k)
        {ts2[j3]=parent1[j];
        j3++;
        }
    }
for(j=0;j<lchrom;j++)
    {newpop[k5].chrom[j]=ts1[j];
    newpop[k5+1].chrom[j]=ts2[j];
    }
}
else
    {for (j=0;j<lchrom;j++)
        {newpop[k5].chrom[j]=parent1[j];
        newpop[k5+1].chrom[j]=parent2[j];
        }
    mutate=flip(pmutation);
    if (mutate)
        {s1=1;
        nmutation=nmutation+1;
        for(j3=0;j3<200;j3++)

```

```

        {j1=random(lchrom);
        j=random(lchrom);
        jj=newpop[k5].chrom[j];
        newpop[k5].chrom[j]=newpop[k5].chrom[j1];
        newpop[k5].chrom[j1]=jj;
        }
    }
mutate=flip(pmutation);
if (mutate)
    {s2=1;
    nmutation=nmutation+1;
    for (j3=0;j3<100;j3++)
        {
            j1=random(lchrom);
            j=random(lchrom);
            jj=newpop[k5+1].chrom[j];
            newpop[k5+1].chrom[j]=newpop[k5+1].chrom[j1];
            newpop[k5+1].chrom[j1]=jj;
        }
    }
}
j2=random(2 * lchrom/3);
for (j=j2;j<j2+lchrom/3-1;j++)
    for (k=0;k<lchrom;k++)
        {if (k==j) continue;
        if (k>j) {i2=k;i1=j;}
        else {i1=k;i2=j;}
        f1=dd[lchrom * newpop[k5].chrom[i1]+newpop[k5].chrom[i2]];
        f1=f1+dd[lchrom * newpop[k5].chrom[(i1+1)%lchrom]
            +newpop[k5].chrom[(i2+1)%lchrom]];
        f2=dd[lchrom * newpop[k5].chrom[i1]
            +newpop[k5].chrom[(i1+1)%lchrom]];

```

```

        f2=f2+dd[lchrom * newpop[k5].chrom[i2]
            +newpop[k5].chrom[(i2+1)%lchrom]];
        if (f1<f2) {inversion(i1,i2,newpop[k5].chrom);}
    }
j2=random(2 * lchrom/3);
for (j=j2;j<j2+lchrom/3-1;j++)
    for(k=0;k<lchrom;k++)
        {if(k==j) continue;
          if (k>j) {i2=k;i1=j;}
                  else {i1=k;i2=j;}
          f1=dd[lchrom * newpop[k5+1].chrom[i1]+newpop[k5+1].chrom
              [i2]];
          f1=f1+dd[lchrom * newpop[k5+1].chrom[(i1+1)%lchrom]
              +newpop[k5+1].chrom[(i2+1)%lchrom]];
          f2=dd[lchrom * newpop[k5+1].chrom[i1]
              +newpop[k5+1].chrom[(i1+1)%lchrom]];
          f2=f2+dd[lchrom * newpop[k5+1].chrom[i2]
              +newpop[k5+1].chrom[(i2+1)%lchrom]];
          if (f1<f2) {inversion(i1,i2,newpop[k5+1].chrom);}
        }
return 1;
}

```

/* 逆转 */

```

void inversion(unsigned int k,unsigned int j,unsigned char * ss)
{unsigned int i,l1;
 unsigned char tt;
 l1=(j-k)/2;
 for (i=0;i<l1;i++)
     {tt=ss[k+i+1];
      ss[k+i+1]=ss[j-i];
      ss[j-i]=tt;
    }
}

```

```

    }
}

/* 重启随机数发生器 */
void randomize1()
{int i;
  randomize();
  for (i=0;i<lchrom;i++)
    oldrand[i]=random(30001)/30000.0;
  jrand=0;
}

```

```

/* 随机数发生器 */
float random1()
{ jrand=jrand+1;
  if (jrand>=lchrom)
    { jrand=0;
      randomize1();
    }
  return oldrand[jrand];
}

```

```

/* 贝努利试验 */
int flip(float probability)
{float ppp;
  ppp=random(20001)/20000.0;
  if(ppp<=probability) return 1;
  return 0;
}

```

附录 C CLS 程序

(1) 该程序是一个简单的概念学习系统,其目的是根据给定的实例学习相应的概念描述。

(2) 运行环境: MS-DOS 6.0, BORLAND C 3.1。

(3) 程序说明: CLS 是由头文件及主程序组成。

1) 数据文件说明: 为了演示需要,为 CLS 配了1个数据文件 4D3C。该文件与一种概念对应,它的每个数据项均为0,1串,与一个实例对应。每个实例有4个属性 F1, F2, F3, F4, 每个属性有4种可能取值 d1, d2, d3, d4, 再加上一个分类位,总计17位。该概念用析取范式可描述如下:

$$4d3c == d1 \vee d2 \vee d3 \vee d4$$

$$d1 == (F1=v1) \& (F2=v1) \& (F3=v1)$$

$$d2 == (F1=v2) \& (F2=v2) \& (F3=v2)$$

$$d3 == (F1=v3) \& (F2=v3) \& (F3=v3)$$

$$d4 == (F1=v4) \& (F2=v4) \& (F3=v4)$$

2) 主要模块说明:

initdata 初始化群体;

generation 采用赌轮法选择、交叉和变异生成新的群体;

checkvalid 检查染色体的有效性,保证每个属性对应的所有位不能全为“0”,否则将其中的任一位置“1”; replace 若两个父代染色体通过遗传操作生成的两个子代中有一个染色体的适应度高于父代中较差的,则用子代中适应度高的去替换父代中较差的染色体。

3) 输入方式有两种: CLS 4D3C nchrom。其中4D3C 为数据文件, nchrom 为群体规模。另一种为交互式。

输出方式: 每运行一代,向文件 4d3c.dat 输出一个该代最大的

适应度,同时通过屏幕输出该代所有染色体及相应的适应度。

CLS 程序清单:

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include <dos.h>
#include "scs2.h"

int          maxrule;
int          chromnum;
float        pcross = 0.6, pmutation = 0.033, sumfitness;
float        avg, max, min;
Chromosome  * chrom, * newchrom;
FILE         * fp, * fpd;
int          gen = 0, maxgen;
const int rulelen = 17;
const int attrnum = 4;
const int attrlen[attrnum] = { 4,4,4,4 };

int main(int argc, char * argv[])
{
    char filename[12];
    srand(time(NULL));
```

```

if (argc == 2)
{strcpy(filename,argv[1]);
do {
    cout << "Input number of chroms(1.." << MAXCHROM <<
    "): ";
    cin >> chromnum;
    }while(chromnum > MAXCHROM);
}
else if (argc == 3)
{strcpy(filename,argv[1]);
    chromnum = atoi(argv[2]);
}
else
{
    cout << "Input a string like nDmC,such as 1d1c" << endl;
    cout << "n & m is a integer between 1 and 4" << endl;
    cin >> filename;
    do{
        cout << "Input number of chroms(1.." << MAXCHROM
        << "): ";
        cin >> chromnum;
    }while(chromnum > MAXCHROM);
}
assert(strlen(filename) == 4);
if (!isdigit(filename[0]) || !isdigit(filename[2]) ||
    (filename[1] != 'd' && filename[1] != 'D') ||
    (filename[3] != 'c' && filename[3] != 'C'))
{
    cout << "Usage:prein [ndmc] [chromnum]" << endl;
    cout << "such as: prein 1D1C 10" << endl;
    cout << "n & m is a integer between 1 and 4" << endl;
    return 1;
}

```

```

    }
if (! (fp=fopen(filename,"rt")))
{
    cout << "file" << filename << "doesn't exists" << endl;
    return 1;
}
strcat(filename, ".DAT");
if (! (fpd=fopen(filename,"wt")))
{
    cout << "file" << filename << "append error" << endl;
    return 1;
}
cout.setf(ios::left,ios::adjustfield);
cout << "input maximum generation" << endl;
cin >> maxgen;
assert (maxgen > 0);
initdata();
objfunc(chrom);
statistics(chrom);
while(gen < maxgen)
{
    gen++;
    report();
    generation();
    objfunc(newchrom);
    replace();
    objfunc(chrom);
    statistics(chrom);
}
report();
fprintf(fpd, "\n");
delete []chrom;

```

```

        delete []newchrom;
        fclose(fp);
        fclose(fpd);
        return 0;
    }

/* 初始化群体 */
void initdata(void)
{
    int i,j,len;
    maxrule = MAXLEN / rulelen; // maxrule number in a chromosome
    chrom = new Chromosome[chromnum];
    newchrom = new Chromosome[chromnum];
    if (!chrom || !newchrom)
    {
        cout << "no enough memory to run this program" << endl;
        fclose(fp);
        fclose(fpd);
        exit(1);
    }
    for(i=0;i<chromnum;i++){
        chrom[i].Rulenum = rand()%maxrule + 1; // at least 1 rule
        chrom[i].Parent1 = chrom[i].Parent2 = 0;
        len = chrom[i].Rulenum * rulelen;
        for(j=0;j<len;j++){ chrom[i].String[j] = (Allele)(rand()%2);
            checkvalid(chrom,i); // check valid of string,prevent all zero attribute
        }
    }

/* 检查染色体的有效性 */
void checkvalid(Chromosome *checkchrom,int index)
{

```

```

Boolean flag;
int curbit = 0,changebit;
int i,j,k;
for(i=0;i<checkchrom[index].Rulenum;i++)
{
    for(j=0;j<attrnum;j++)
    {
        flag = true;
        for(k=0;k<attrlen[j];k++)
            if (checkchrom[index].String[curbit+k] == true)
            {
                flag = false;
                break;
            }
        if (flag == true){
            changebit = curbit + rand()%attrlen[j];
            checkchrom[index].String[changebit] = true;
        }
        curbit += attrlen[j];
    }
    /* 跳过 class 位 */
    curbit++;
}

/* 从染色体中抽取实例 */
void extractstr(char *str,int curbit,int length,int index)
{
    int i;
    for(i=0;i<length;i++)
        if (chrom[index].String[curbit+i] == false)
            *str = '0';

```

```

        else
            *str = '1';
            str++;
        }
        *str = '\0';
    }

/* 显示染色体及其适应度 */
void report(void)
{
    int i,j,k;
    int curbit;
    char str[20];
    cout << endl;
    for(i=0;i<chromnum;i++)
    {
        curbit = 0;
        for(j=0;j<chrom[i].Rulenum;j++){
            for(k=0;k<attrnum;k++){
                cout << "F" << setw(attrlen[k]) << k+1;
                cout << " class ";
            }

            cout << " fitness" << endl;
            for(j=0;j<chrom[i].Rulenum;j++){
                for(k=0;k<attrnum;k++){
                    extractstr(str,curbit,attrlen[k],i);
                    curbit += attrlen[k];
                    cout << setw(attrlen[k]+1) << str;
                }

                extractstr(str,curbit,1,i);
                curbit++;
                cout << ' ' << setw(6) << str;
            }
        }
    }
}

```

```

        cout << ' ' << chrom[i].Fitness;
        cout << endl << endl;
    }
    cout << "max = " << max << endl;
    cout << "avg = " << avg << endl;
    cout << "min = " << min << endl;
    cout << endl << endl;
}

/* 统计群体的最大、最小、平均适应度 */
void statistics(Chromosome * pop)
{
    int j,maxindex = 0;
    min = max = sumfitness = pop[0].Fitness;
    for (j=1;j<chromnum;j++) // Loop for max,min,sumfitness
        {sumfitness += pop[j].Fitness;
        if (pop[j].Fitness > max)
            {max = pop[j].Fitness;
            maxindex = j;
            }
        if (pop[j].Fitness < min) min = pop[j].Fitness;
        }
    avg = sumfitness/chromnum;
    fprintf(fpd,"%f\n",chrom[maxindex].Fitness);
}

/* 生成随机数0,1 */
inline float random01(void)
{
    return rnd(0,1);
}

```

```

/* 生成范围在 low 与 high 之间的一个随机数 */
float rnd(int low,int high)
{
    int i;
    float j;
    i = rand();
    j = low + (high-low) * float(i) / float(RAND_MAX);
    return j;
}

Boolean flip(float val)
{
    float j = rnd(0,1);
    if (j <= val) return true;
    else return false;
}

/* 生成下一代群体 */
void generation(void)
{
    // Create a new generation through ; select , crossover , and mutation
    int j,mate1,mate2;
    int even_number = (chromnum%2) ? (chromnum-1) : chromnum;
    // If chrom number is a odd number,then the last don't crossover
    j = 0;
    while (j < even_number)
    {
        mate1 = select(); // Pick pair of mates
        mate2 = select();
        crossover(mate1,mate2,j);
        j += 2;
    }
    if (even_number != chromnum)
        • 424 •

```



```

        {for(mate1=0;mate1<chrom[j].Rulenum*rulelen;mate1++)
            newchrom[j].String[mate1] = mutation(chrom[j].String[mate1]);
            newchrom[j].Rulenum = chrom[j].Rulenum;
            newchrom[j].Parent1 = newchrom[j].Parent2 = j;
            checkvalid(newchrom,j);
        }
    }
}

```

```

int select(void)
{
    float rand,partsum=0.0;
    int j=-1;
    rand = random01()*sumfitness;
    do {
        j++;
        partsum += chrom[j].Fitness;
    }while((partsum < rand) && (j!=chromnum-1));
    return j; // Return individual number
}

```

```

Allele mutation(Allele alleleval)
{
    // Mutate an allele w/pmutation,count number of mutations
    Boolean mutate;
    mutate = flip(pmutation);
    if (mutate==true)
        return (Allele)(!alleleval);
    else
        return alleleval;
}

```

```

void crossover(int parent1,int parent2,int child)

```

```

{
// Cross 2 parent strings, place in 2 child strings
int r1,r2,b,i;
int len1,len2;
Boolean flag = true;
r1 = rand()%chrom[parent1].Rulenum;
r2 = rand()%chrom[parent2].Rulenum;
b = rand()%rulelen;
len1 = r1 + chrom[parent2].Rulenum - r2;
len2 = r2 + chrom[parent1].Rulenum - r1;
if (len1 > maxrule || len2 > maxrule)
    flag = false;
if ((flag == true) && flip(pcross))
{
newchrom[child].Rulenum = len1;
newchrom[child+1].Rulenum = len2;
newchrom[child].Parent1 = newchrom[child+1].Parent1 = parent1;
newchrom[child].Parent2 = newchrom[child+1].Parent2 = parent2;
len1 = r1 * rulelen + b;
len2 = chrom[parent2].Rulenum * rulelen - r2 * rulelen - b;
for(i=0;i<len1;i++)
    newchrom[child].String[i] = mutation(chrom[parent1].String[i]);
for(i=0;i<len2;i++)
    newchrom[child].String[len1+i] = /* 后接下一句 */
        mutation(chrom[parent2].String[r2 * rulelen + b + i]);
len1 = r2 * rulelen + b;
len2 = chrom[parent1].Rulenum * rulelen - r1 * rulelen - b;
for(i=0;i<len1;i++)
    newchrom[child+1].String[i] = mutation(chrom[parent2].String[i]);
for(i=0;i<len2;i++)
    newchrom[child+1].String[len1+i] = /* 后接下一句 */
        mutation(chrom[parent1].String[r1 * rulelen + b + i]);
}
}

```

```

else{
    newchrom[child ].Rulenum = chrom[parent1].Rulenum;
    newchrom[child+1].Rulenum = chrom[parent2].Rulenum;
    newchrom[child].Parent1 = newchrom[child].Parent2 = parent1;
    newchrom[child+1].Parent1 = newchrom[child+1].Parent2 =
    parent2;
    for(i=0;i<chrom[parent1].Rulenum * rulelen;i++)
        newchrom[child ].String[i] = mutation(chrom[parent1].String
        [i]);
    for(i=0;i<chrom[parent2].Rulenum * rulelen;i++)
        newchrom [child + 1].String[i] = mutation(chrom [parent2].
        String[i]);
    }
    checkvalid(newchrom,child);
    checkvalid(newchrom,child+1);
}

```

/* 适应度 = 正确分类数 / 实例总数 (= 256) * /

```

void objfunc(Chromosome * pop)
{
    const int len = rulelen + 1;
    char str[len];
    int fi[MAXCHROM];
    int i,j,k;
    Allele flag;
    int bitoftrue;
    for(i=0;i<chromnum;i++) fi[i] = 0;
    while(fread(str,sizeof(char),len,fp) == len)
    {
        for(i=0;i<chromnum;i++){
            for(j=0;j<pop[i].Rulenum;j++){
                bitoftrue = 0;

```

```

    flag = true;
    for(k=0;k<attrnum;k++){
        while(str[bitoftrue] == '0')
            bitoftrue++;
        if(pop[i].String[j * rulelen + bitoftrue] == false){
            flag = false;
            break;
        }

        else bitoftrue++;
    }

    if ((flag == true) && (str[rulelen-1] == '0') == /* 后接下一句 */
        (int)(pop[i].String[j * rulelen + rulelen - 1])){
        fi[i]++;
        break;
    } /* if */

    } /* j , Rulenum */

    } /* i , chromnum */

} /* fread */
for(i=0;i<chromnum;i++){
    pop[i].Fitness = float(fi[i])/256.0; // total examples
    rewind(fp);
}

```

/* 若两个父代染色体通过遗传操作生成的两个子代中有一个染色体的适应度高于父代中较差的,则用子代中适应度高的去替换父代中较差的染色体 */

```

void replace(void)
{
    int even_number = (chromnum%2) ? (chromnum-1) : chromnum;
    int i = 0;
    Boolean flag;
    int maxofchild,minofparent;

```

```

while(i<even_number){
    maxofchild = (newchrom[i].Fitness > newchrom[i+1].Fitness) ? i : i+
    1;
    minofparent = (chrom[newchrom[i].Parent1].Fitness<
    chrom [newchrom [i]. Parent2 ]. Fitness ) ? newchrom [i]. Parent1 :
    newchrom[i].Parent2;
    if (newchrom[maxofchild].Fitness > chrom[minofparent].Fitness)
    {
        chrom[minofparent].Rulenum = newchrom[maxofchild].Rulenum;
        for(int j=0;j<newchrom[maxofchild].Rulenum * rulelen;j++)
            chrom[minofparent].String[j] = newchrom[maxofchild].String[j];
    }
    i += 2;
}

if (even_number != chromnum){
    if (newchrom[i].Fitness > chrom[newchrom[i].Parent1].Fitness){
        chrom[newchrom[i].Parent1].Rulenum = newchrom[i].Rulenum;
        for(int j=0;j<newchrom[i].Rulenum * rulelen;j++)
            chrom [newchrom [i]. Parent1 ]. String [j] = newchrom [i]. String
            [j];
    }
}
}

```

头文件：

```

const int MAXLEN = 60;           // max length of a chromosome
const int MAXCHROM = 100;       // max chroms (max popsize)
const float MAXFITNESS = 0.85; // fitness accepted
enum tagboolean {false=0,true=1};
typedef enum tagboolean Allele;
typedef enum tagboolean Boolean;
typedef struct {
    int Rulenum;

```

```

        Allele String[MAXLEN];
        float Fitness;
        int Parent1,Parent2;
    }Chromosome;

void initdata(void);
void checkvalid(Chromosome * checkchrom,int index);
void extractstr(char * str,int curbit,int length,int index);
void report(void);
void statistics(Chromosome * pop);
int select(void);
Allele mutation(Allele alleleval);
void crossover(int parent1,int parent2,int child);
void objfunc(Chromosome * pop);
void replace(void);
void generation(void);
inline float random01(void);
float rnd(int low,int high);
Boolean flip(float val);
void reportgraph(char * ,int);
void initcoords(int);

```

| | | |
|-------------------|-------------------|-------------------|
| 00010001000100011 | 00010001010010000 | 00010010001001000 |
| 00010001000100101 | 00010001100000010 | 00010010001010000 |
| 00010001000101001 | 00010001100000100 | 00010010010000010 |
| 00010001000110001 | 00010001100001000 | 00010010010000100 |
| 00010001001000010 | 00010001100010000 | 00010010010001000 |
| 00010001001000100 | 00010010000100010 | 00010010010010000 |
| 00010001001001000 | 00010010000100100 | 00010010100000010 |
| 00010001001010000 | 00010010000101000 | 00010010100000100 |
| 00010001010000010 | 00010010000110000 | 00010010100001000 |
| 00010001010000100 | 00010010001000010 | 00010010100010000 |
| 00010001010001000 | 00010010001000100 | 00010100000100010 |

| | | |
|---------------------|--------------------|---------------------|
| 00010100000100100 | 00100001000100100 | 00100100000100100 |
| 00010100000101000 | 00100001000101000 | 00100100000101000 |
| 00010100000110000 | 00100001000110000 | 00100100000110000 |
| 00010100001000010 | 00100001001000010 | 00100100001000010 |
| 00010100001000100 | 00100001001000100 | 00100100001000100 |
| 00010100001001000 | 00100001001001000 | 00100100001001000 |
| 00010100001010000 | 00100001001010000 | 00100100001010000 |
| 00010100010000010 | 00100001010000010 | 00100100010000010 |
| 00010100010000100 | 00100001010000100 | 00100100010000100 |
| 00010100010001000 | 00100001010001000 | 00100100010001000 |
| 00010100010010000 | 00100001010010000 | 00100100010010000 |
| 00010100010010000 | 00100001010010000 | 00100100010010000 |
| 00010100010010000 | 00100001010010000 | 00100100010010000 |
| 000101000100000010 | 00100001100000010 | 001001000100000010 |
| 0001010001000000100 | 001000011000000100 | 0010010001000000100 |
| 0001010001000001000 | 001000011000001000 | 0010010001000001000 |
| 0001010001000100000 | 00100001100010000 | 0010010001000100000 |
| 00011000000100010 | 00100010000100010 | 00101000000100010 |
| 00011000000100100 | 00100010000100100 | 00101000000100100 |
| 00011000000101000 | 00100010000101000 | 00101000000101000 |
| 00011000000110000 | 00100010000110000 | 00101000000110000 |
| 00011000001000010 | 00100010001000011 | 00101000001000010 |
| 00011000001000100 | 00100010001000101 | 00101000001000100 |
| 00011000001001000 | 00100010001001001 | 00101000001001000 |
| 00011000001010000 | 00100010001010001 | 00101000001010000 |
| 00011000010000010 | 00100010010000010 | 00101000010000010 |
| 00011000010000100 | 00100010010000100 | 00101000010000100 |
| 00011000010001000 | 00100010010001000 | 00101000010001000 |
| 00011000010010000 | 00100010010010000 | 00101000010010000 |
| 00011000100000010 | 00100010100000010 | 00101000100000010 |
| 00011000100000100 | 00100010100000100 | 00101000100000100 |
| 00011000100001000 | 00100010100001000 | 00101000100001000 |
| 00011000100001000 | 00100010100001000 | 00101000100001000 |
| 00011000100010000 | 00100010100010000 | 00101000100010000 |
| 00011000100000010 | 00100010100000010 | 00101000100000010 |
| 00011000100000100 | 00100010100000100 | 00101000100000100 |
| 00011000100001000 | 00100010100001000 | 00101000100001000 |
| 00011000100010000 | 00100010100010000 | 00101000100010000 |
| 00100001000100010 | 00100100000100010 | 01000001000100010 |

| | | |
|-------------------|-------------------|-------------------|
| 01000001000100100 | 01000100000100100 | 10000001000100100 |
| 01000001000101000 | 01000100000101000 | 10000001000101000 |
| 01000001000110000 | 01000100000110000 | 10000001000110000 |
| 01000001001000010 | 01000100001000010 | 10000001001000010 |
| 01000001001000100 | 01000100001000100 | 10000001001000100 |
| 01000001001001000 | 01000100001001000 | 10000001001001000 |
| 01000001001010000 | 01000100001010000 | 10000001001010000 |
| 01000001010000010 | 01000100010000011 | 10000001010000010 |
| 01000001010000100 | 01000100010000101 | 10000001010000100 |
| 01000001010001000 | 01000100010001001 | 10000001010001000 |
| 01000001010010000 | 01000100010010001 | 10000001010010000 |
| 01000001100000010 | 01000100100000010 | 10000001100000010 |
| 01000001100000100 | 01000100100000100 | 10000001100000100 |
| 01000001100001000 | 01000100100001000 | 10000001100001000 |
| 01000001100010000 | 01000100100010000 | 10000001100010000 |
| 01000010000100010 | 01001000000100010 | 10000010000100010 |
| 01000010000100100 | 01001000000100100 | 10000010000100100 |
| 01000010000101000 | 01001000000101000 | 10000010000101000 |
| 01000010000110000 | 01001000000110000 | 10000010000110000 |
| 01000010001000010 | 01001000001000010 | 10000010001000010 |
| 01000010001000100 | 01001000001000100 | 10000010001000100 |
| 01000010001001000 | 01001000001001000 | 10000010001001000 |
| 01000010001010000 | 01001000001010000 | 10000010001010000 |
| 01000010010000010 | 01001000010000010 | 10000010010000010 |
| 01000010001000100 | 01001000010001000 | 10000010001000100 |
| 01000010001010000 | 01001000010010000 | 10000010001010000 |
| 01000010010000010 | 01001000010000010 | 10000010010000010 |
| 01000010010000100 | 01001000010000100 | 10000010010000100 |
| 01000010010001000 | 01001000010001000 | 10000010010001000 |
| 01000010010010000 | 01001000010010000 | 10000010010010000 |
| 01000010100000010 | 01001000100000010 | 10000010100000010 |
| 01000010100000100 | 01001000100000100 | 10000010100000100 |
| 01000010100001000 | 01001000100001000 | 10000010100001000 |
| 01000010100010000 | 01001000100010000 | 10000010100010000 |
| 01000100000100010 | 10000001000100010 | 10000100000100010 |

10000100000100100
10000100000101000
10000100000110000
10000100001000010
10000100001000100
10000100001001000
10000100001010000
10000100010000010
10000100010000100
10000100010001000
10000100010010000
10000100100000010
10000100100000100
10000100100001000
10000100100010000
10001000000100010
10001000000100100
10001000000101000
10001000000110000
10001000001000010
10001000001000100
10001000001001000
10001000001010000
10001000010000010
10001000010000100
10001000010001000
10001000010010000
10001000100000011
10001000100000101
10001000100001001
10001000100010001